НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

«До захисту допущено»
в. о. завідувача кафедри
_____ О.Л. Тимощук
«___»_____20__ р.

Дипломна робота
на здобуття ступеня бакалавра
з напряму підготовки 6.040303 "Системний аналіз"
на тему: «Адаптивні засоби захисту комп'ютерних систем на основі апарата нейронних мереж»

Виконав:
студент IV курсу, групи КА-64
Кунтиш Олексій Сергійович _____
Керівник:
Професор Мухін В.Є. _____

Консультант з економічного розділу:
доцент, к.е.н. Шевчук О.А. _____

Консультант з нормоконтролю:
доцент, к.т.н. Коваленко А.Є. _____

Рецензент:
доцент, к.т.н., доцент кафедри технічної кібернетики,
Корнага Ярослав Ігорович _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших
авторів без відповідних посилань
Студент _____

Київ – 2020

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 124 «Системний аналіз»

Освітньо-професійна програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ

В.о.завідувача кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Кунтиш Олексій Сергійович

1. Тема роботи «Адаптивні засоби захисту комп'ютерних систем на основі апарата штучних нейронних мереж», керівник роботи професор кафедри ММСА, д.т.н. Мухін В.Є., затверджені наказом по університету від «_25_»_травня_ 20_20_ р. №_1143-с__

2. Термін подання студентом роботи 08 травня 2020 року

3. Вихідні дані до роботи

1. Операційна система Windows
2. Частота процесора 2.7 ГГц
3. Мова програмування Python 3
4. Середовище розробки – Jupyter Notebook
5. Бібліотеки, що використовувалися: Pandas, Sklearn, NumPy

4. Зміст роботи

1. Проаналізувати існуючі математичні моделі та готові програмні продукти
2. Проаналізувати доцыльныcть використання нейронних мереж в заданій області
3. Розробити програмний продукт виявлення загроз у комп'ютерних систем

4. Розробити систему оцінки побудованої моделі
5. Виконати економічний аналіз програмного продукту

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

1. Презентація

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	к.е.н., доц. Шевчук О. А.	21.04.20	28.05.20

7. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	13.04.20	
2	Аналіз існуючих рішень	22.04.20	
3	Підбір алгоритму для реалізації	28.04.20	
4	Розробка продукту для моделювання та системи оцінювання моделі	12.05.20	

* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи.

5	Тестування продукту, отримання результатів	15.05.20	
6	Оформлення дипломної роботи	18.05.20	

Студент Кунтиш О.С.

Керівник Мухін В.Є.

РЕФЕРАТ

Дипломна робота: 103 с., 6 табл., 2 дод, 11 рис, 10 джерел.

АДАПТИВНІ ЗАСОБИ ЗАХИСТУ КОМП'ЮТЕРНИХ СИСТЕМ НА ОСНОВІ АПАРАТА ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ.

Метою даної роботи є побудова модуля, що базуватиметься на основі нейронних мереж, та буде на основі вхідних даних виявляти загрози комп'ютерної системи.

Актуальність теми: в наш час дуже актуальною є тема захисту комп'ютерних систем. У всіх сферах своєї життєдільності людина працює з різного виду комп'ютерними системами, а їх захищеніст дозволяє гарантувати конфіденційну й безперервну роботу.

Предметом дослідження є програмний модуль, що побудований на основі зростаючого нейронного газу.

Об'єктом досліджень є підготована за певними стандартами вибірка, котра складається з даних щодо трафіку певної комп'ютерної системи.

Метод дослідження: застосований зростаючий нейронний газ та супутні алгоритми навчання мережі були виконані на мові програмування Python 3.

Отримані результати: Була побудована та навчена штучна нейрона мережа. Далі було оцінено роботу алгоритму на тестовій вибірці, апроведений аналіз точності роботи програмного модуля і зроблені висновки щодо доцільності використання цього алгоритму задля захисту комп'ютерних систем.

ABSTRACT

Thesis: 93 pp., 6 tables, 2 appendices, 42 figs., 10 referencs

ADAPTIVE MEANS OF PROTECTION OF COMPUTER SYSTEMS BASED ON THE DEVICE OF ARTIFICIAL NEURAL NETWORKS.

The purpose of this work is to build a module that will be based on neural networks, and will be based on input data to detect threats to the computer system.

Relevance of the topic: nowadays the topic of computer systems protection is very relevant. In all spheres of life, people work with different types of computer systems, and their security allows you to guarantee confidential and continuous work.

The subject of the study is a software module based on growing neural gas.

The object of research is a sample prepared according to certain standards, which consists of data on the traffic of a particular computer system.

Research method: the applied neural gas and related network learning algorithms were performed in the Python 3 programming language.

Results: An artificial neural network was built and trained. Next, the operation of the algorithm on the test sample was evaluated, the analysis of the accuracy of the software module was performed and conclusions were made about the expediency of using this algorithm to protect computer systems.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

IDS – Intrusion Detection System

ІБ - інформаційна безпека

IT - Information Technology

БД – база даних

КС – комп'ютерна система

НМ – нейрона мережа

ВСТУП

В наш час цифрових технологій комп’ютер є невід’ємною складовою як повсякденного життя людини, так і вагомою складовою робочого процесу майже будь-якої компанії. Тому питання про захищеність комп’ютерних систем завжди є дуже актуальним.

З невпинним розвитком технологій зростає складність та ієрархічність комп’ютерних систем, тому стає дедалі складніше відслідити їх рівень захищеності вцілому. Для цього розроблено багато методик, методів, принципів захисту комп’ютерних систем. З їх допомогою, людина, що має таку потребу, може виявити слабкі місця певною комп’ютерної системи, оцінити рівень захищеності, а також нейтралізувати загрозу. При наявності даної інформації легше протидіяти загрозам та вторгненням у комп’ютерні системи.

Мета цієї роботи - є вибір або розробка оптимального програмного продукту для розпізнавання загроз з використанням механізму штучних нейронних мереж, також глибокий аналіз та порівняння результатів, наведення висновків.

Результатом даної роботи є вихідні данні програмного продукту, що базуються на основі певної моделі даних, висновки щодо роботи програмного

INTRODUCTION

In today's digital technology, the computer is an integral part of both everyday life and an important part of the work process of almost any company. Therefore, the issue of security of computer systems is always very important.

With the constant development of technology, the complexity and hierarchy of computer systems is growing, so it is becoming increasingly difficult to track their level of security as a whole. Many methods, techniques, principles of protection of computer systems have been developed for this purpose. With their help, a person in need can identify weaknesses in a particular computer system, assess the level of security, and neutralize the threat. With this information, it is easier to counter threats and intrusions into computer systems.

The purpose of this work is to select or develop the optimal software product for threat recognition using the mechanism of artificial neural networks, as well as in-depth analysis and comparison of results, drawing conclusions.

The result of this work is the initial data of the software product, based on a certain data model, conclusions about the operation of the software product, general analysis.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ, ПОРІВНЯЛЬНИЙ АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

1.1 Загальне поняття комп'ютерної системи, захищеності комп'ютерної системи.

Для оперування терміном безпека комп'ютерної системи, опишемо ряд визначень, що стосуються даної предметної області.

Сучасні комп'ютерні системи – складний механізм, складовими якого є багато компонентів різного рівня автоматизованості, пов'язані один з одним, та дані, якими вони обмінюються.

Безпека системи – набір заходів, що мають на меті захистити систему від випадкового або навмисного втручання у її роботу [1].

Загроза безпеці - потенційний вплив на комп'ютерну систему, що може прямо чи периферійно змінити роботу комп'ютерної системи і завдати шкоду її власникам чи користувачам.

Атака (вторгнення) – приведення загрози в дію.

Система виявлення атак(вторгнень) - (англ. *Intrusion Detection System, IDS*) — програмний або апаратний засіб, призначений для виявлення фактів несанкціонованого доступу в комп'ютерну систему або мережу або несанкціонованого управління ними в основному через Інтернет [2].

1.2 Класифікація систем захисту комп'ютерних мереж

Задля забезпечення максимального рівня захисту комп'ютерна мережа має бути інформованою щодо можливих атак задля їх нейтралізації. Основним засобом ідентифікації атак є IDS(англ. Система ідентифікації вторгнень) [3].

Існує декілька типів систем ідентифікації вторгнень. Зазвичай, їх поділяють на:

– системи рівня мережі;

– системи рівня вузла, тобто ті, що ідентифікують зміни на окремому елементі системи;

– системи, що базуються на оцінці вразливостей.

Цю класифікацію можна розширити. Для цього опишемо задачі IDS.

До задач IDS відносяться:

– отримання даних;

– інтерпретація отриманих даних;

– представлення результатів.

Відповідно до цих задач, всі системи можна класифікувати по цим параметрам:

– тип даних, що збираються;

– метод отримання даних;

– метод інтерпретації даних;

– метод представлення результату.

Система виявлення загроз за вхідні данні може мати:

– дані, що збираються про окремий вузол мережі;

– дані, що збираються по всій мережі в цілому;

– змішаний тип даних.

Дані, що збираються про конкретиний вузол мережі – лише ті дані, що стосуються одного вузла і частково тих, що знаходяться з ним у взаємодії.

Дані, що збираються по всій системі – загальна картина мережевої взаємодії.

Зазвичай, системи виявлення атак не збирають дані такого типу, оскільки це ресурсозатратно.

Тип отримання данних IDS можна поділити на:

– пасивний;

– активний;

– змішаний.

Пасивне отримання даних базується на простому спостеріганні за комп’ютерною системою, не впливаючи на роботу мережі. Використовуючи такий метод, IDS виконує лише аналіз трафіка. Найчастіше використовується у IDS [4].

Активний метод базується на прямій взаємодії з комп’ютерною мережею. Використовується рідше, так як несе додаткове навантаження на комп’ютерну систему, хоча і має ряд переваг, наприклад, превентивна ідентифікація загроз.

Методи інтерпретації даних можна поділити на:

– методи виявлення відомих загроз;

– методи виявлення аномалій;

Методи інтерпретації даних:

– булевий. Система відповідає «так» або «ні» на питання «Чи є проблемиу мережі?»;

– об’єктний. Система відповідає на питання «Які зараз є проблеми».

Незалежно від класифікації, типова IDS складається з наступних компонентів:

– Датчики;

1. модулі захоплення;
2. модулі збору;

3. модулі сполучення.

– Підсистема аналізу;

1. модулі декодування;

2. модулі виявлення.

– Підсистема виводу.

1. модулі реагування;

2. модулі виводу.

Датчиків в сучасних IDS, як правило кілька, і вони можуть бути розподілені по машинам користувача. Завданням датчиків є захоплення даних, їх накопичення і відправка до центральної системи.

Датчики поділяють на:

– датчики додатків - надають дані про роботу конкретного програмного забезпечення, що захищається системи;

– датчики вузла - надають дані про функціонування окремої робочої станції системи, що захищається;

– датчики мережі - відповідають за збір даних для оцінки мережевого трафіку;

– міжмережеві датчики - містять характеристики даних, що циркулюють між мережами.

Отримана від датчиків інформація декодується в набір ознак і проводиться через ланцюжок модулів аналізу. Модуль аналізу приймає інформацію з джерела даних і аналізує дані на наявність ознак атак або порушень політики безпеки. Він може використовуватися, як для виявлення вторгнень, що експлуатують відомі шаблони атак, так і для виявлення аномальної активності. Аналізатори відомих атак можуть спиратися на оновлювану базу сигнатур, або бути реалізовані на основі мережі прямого

поширення, навченої на прикладах атак. Аналізатори аномалій можуть враховувати не тільки стан датчиків, а й результати роботи попередніх аналізаторів. Після того, як аналізатори відпрацювали, в тому чи іншому вигляді виводиться їх сукупний статок. Це може бути, як висновок даних користувачеві через один з інтерфейсів, за що відповідають модулі виведення, так і вказівку мети активної системі захисту (наприклад, брандмауер на блокування з'єднання), за що відповідають модулі реагування [5].

Приблизну схему роботи представлено на рисунку 1.

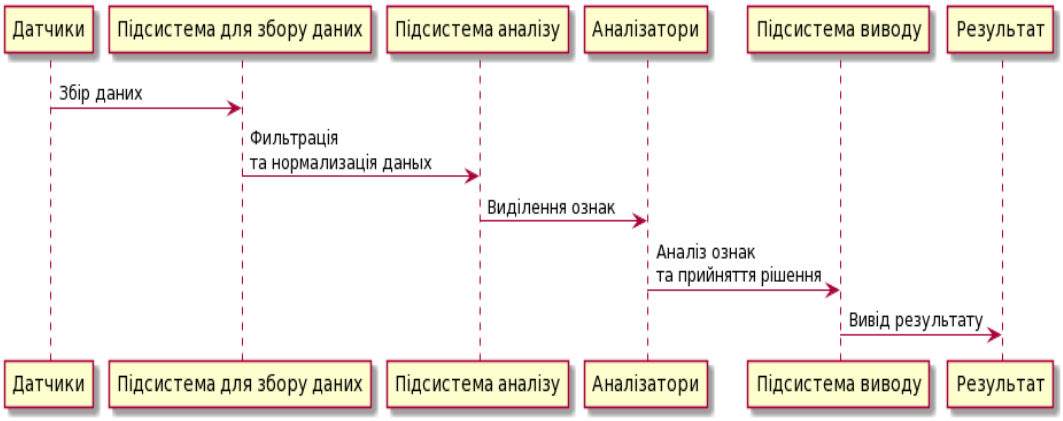

Рисунок 1.1 - Приблизна схема роботи системи захисту комп’ютерних мереж

1.3 Порівняльний аналіз існуючих рішень

На сьогоднішній день в даній фері існують готові продукти для вирішення заданої проблеми. У цьому розділі розглянемо деякі з них.

1. Snort та Suricata

Ці дві IDS об’єднує те, що вони обидві є системами рівня мережі за стандартною моделлю OSI. Далі наведено особливості кожної з них.

Snort – звичайна IDS мережевого рівня, що працює за принципом аналізу трафіку з певною скомпонованою базою правил. Тобто, фактично Snort намагається виявити вже відомі порушення. Але, оскільки Snort є рішенням з відкритим кодом, то на його базі створено багато надлаштувань та нових реалізацій. Для Snort можна написати власний модуль для виявленя аномалій у трафіку, що і було зроблено багатьма навіть на комерційній основі. Можливо навіть відступлення від основного принципу роботи. Так, вже є рішення, що доповнюють Snort і порівнюють трафік не з певними, завчасно скомпонованими правилами, а мають у своєму складі окремий модуль для виявлення аномального трафіку.

Suricata має правила, цілком сумісні з системою Snort. Але, навідміну від Snort, проводить оцінку трафіка не лише на мережевому рівні, але й на рівні роботи прикладних протоколів. Також є можливість для написання правил оцінки трафіку на мові Lua, що дає більше можливостей оцінки мережі. Також Suricata має змогу проводити аналіз цілого трафіку між певними вузлами системи, а не аналізувати лише певні пакети, що збільшує якість виявлення загроз. Таклж Suricata адаптується і навчається в певній мережі за час роботи з нею [6].

2. Система Bro

Дана система також працює на рівні мережі, як і дві попередні. Основний принцип роботи – пошук завчасно відомих порушень, проте може відслідковувати досі невідому аномальну активність. Це досягається завдяки двум факторам:

– Дані, що приймає система, попередньо перевіряються на незвичність.

– Дані порівнюються з завчасно описаною схемою роботи даної мережі

Bro, навідміну від двох попередніх систем, аналізує не сам трафік, а виконує отримані пакети на віртуальній машині, перетворюючи їх на події більш високого рівня, які далі аналізуються в системі Bro [7].

На рисунку 1 зображена приблизна схема роботи системи Bro.

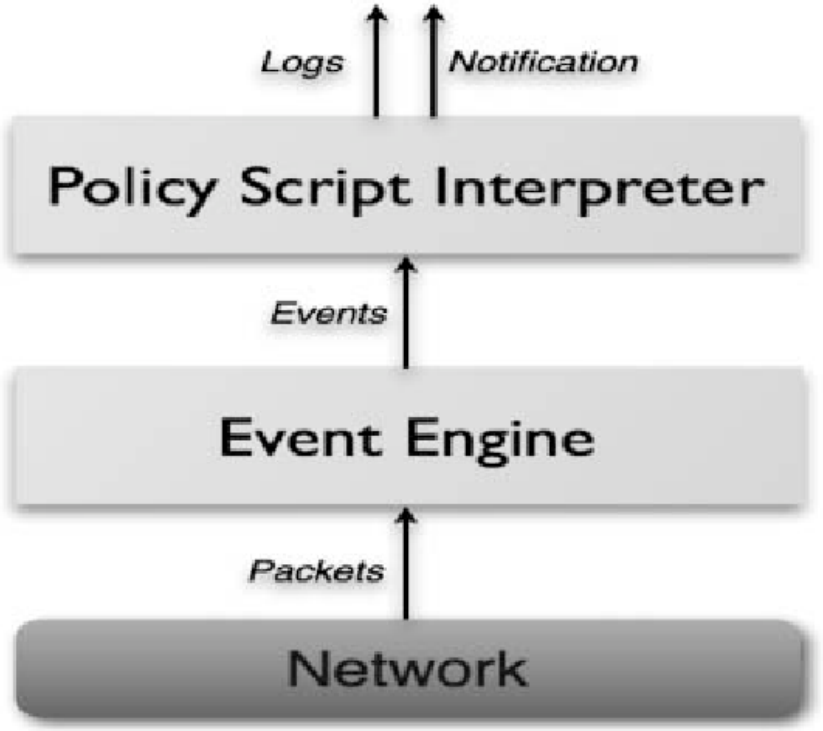

Рисунок 1.2 - Приблизна схема роботи системи Bro

1.4 Алгоритми для виявлення загроз в IDS

На сьогоднышний день системи IDS доволі широко використовуються для безпечної роботи комп'ютерних систем. Тому існує широкий спектр алгоритмів, що реалізовані в системах захисту. Нижче наведемо деякі з них.

– аналіз на основі предписаних правил;

– статистичні системи;

– аналіз по сигнатурам.

Всі ці алгоритми мають суттєві недоліки:

– аналіз по сигнатурам не буде реагувати на невідому йому атаку. Достатньо невеликих коректив сигнатури атаки для того, щоб система більше не розпізнавала цю атаку;

– системи, що базуються на правилах, зазвичай не мають достатньої гнучкості для задання правил, що ставить рамки на можливі загрози, що будуть виявлені. Також, незначні зміни в послідовності дій при атаці можуть повливати на детектор так, що він буде не взмозі виявити цю атаку. Також, для нормальної роботи систем цього виду необхідно постійно оновлувати актуальність бази предписаних правил;

– порушники за певний період часу можуть перенавчити систему так, що аномальна активність буде сприйматися як нормальна.

Ще одним алгоритмом для виявлення атак є нейронні мережі. Нейронні мережі не мають такого широкого застосунку, як попередньо представлені методи, але є більш функціональними та перспективними. Алгоритми, основані на основі нейронних мереж, мають ряд перевег над іншими:

– НМ мають можливість до аналізу неповних даних, а також вони здатні аналізувати сигнали з шумом;

– відсутня необхідність до формалізації знань;

– руйнування окремих елементів або зв’язків НМ не означає про повний вихід із ладу всього аналізатору;

– існує можливість щодо виявлення невідомих атак;

– НМ здатні до навчання в процесі роботи;

– НМ потребують меншого втручання людини;

– при використанні нейронних мереж зростає швидкысть роботи аналізатора, адже НМ дають змогу паралелізувати роботу [8].

Звичайно, системи, що базуються на основі нейронних мереж, мають і свої недоліки. Наприклад, рішення, що прийняла нейронна мережа в тій чи іншій ситуаціїї, дуже складно обгрунтувати. Також, перед початком роботи аналізаторів даного типу, нейронна мережа потребує навчання і тестування, що несе за собою витрати в часі та ресурсах. До того ж, ці процеси відбуваються за умови наявності тестової та тренувальної вибірок, які не завжди просто сформувати.

1.5 Висновки

У цьому розділі було розглянуто поняття захищеності комп’ютерних систем та можливості щодо їх захисту. Також була проаналізована інформація щодо існуючих систем та їх розбиття на категорії. У третьому підрозділі були проаналізовані існуючі реалізації систем захисту комп’ютерних систем, а у 4 – методи, які вони використовують задля виявлення загроз.

2 ОПИС ВИБРАНОЇ МОДЕЛІ

2.1 Доречність використання НМ для оцінки вторгнень

Проаналізувавши теоретичні роботи було зроблено висновок, що у загальному випадку КС доречність використання нейромережевих методів розпізнавання позиціонується такими рисами:

– малою ефективністю або відстутністю алгоритмів та методів розпізнавання;

– можливістю навчання НМ;

– можливістю забезпечення технічних аспектів використання НМ.

Для того, щоб результативно навчити НМ, необхідно слідкувати певним умовам, що приведені нижче:

– в піддослідному процесі вияивити кількісні показники, які будуть інформативними та використаються у якості вхідних та вихідних даних НМ.

– сформувати навчальну вибірку НМ;

– під час роботи з вказаним обсягом обчислювальних ресурсів та допустимій похибці під час навчання, строк навчання НМ не має бути більшим за вказаний інтервал часу [9].

Поглибимось у вказані вище умови з погляду використання НМ під час використання базових методів для виявлення атак – зловживань та аномалій. Необхідно вказати, що якщо використовувати метод зловживань в IDS, НМ може розпізнавати сигнатуру атак (СА), іншими словами величини даних, які контролюються і вказують на створення атаки. Під час використання методу аномалій НМ даний метод працює хз метою виявлення відхилення параметрів, що контролюються від шаблонів нормальної поведінки (ШНП) ресурсів КС.

Вхідні параметри НМ можуть бути представлені у вигляді зареєстрованих на експлуатації функціональних параметрів КС, які застосовуються для розпізнавання атак. Перед тим, як відправити в НМ певні функціональні параметри необхідно спеціально привести їх у нормалізований стан та закодувати. Якщо перед нами самий простий випадок, тоді на вихід НМ подаватиметься інформація про присутність або відсутність атак.

У більш складних ситуаціях вихідні дані НМ повинні включати у себе ще й тип атаки. Згідно цієї інформації, для організації навчальних прикладів потребуються дані на основі статистичного аналізу щодо величин функціональних параметрів при здійсненні атаки та коли НМ працює у нормальному стані. У навчальній вибірці необхідно у повному обсязі показати усі види атак, які можуть відбутися та профілі під час нормальної поведінки елементів КС. Згідно з вищесказаного множина навчальних прикладів розраховується за допомогою формули 1:

$$V_A \bigcup V_N \in O, \qquad (2.1)$$

де O – множина навчальних прикладів, V_A – множина образів можливих атак, V_N – множина профілів нормальної поведінки.

Також під час створення та реалізації навчальної вибірки необхідно зважати на вказані нижче вимоги;

1. мережа, яка проходила навчання на прикладах нормальної/аномальної роботи певної КС має можливість показувати невірний результат для інших КС;

2. вставка або видалення з КС деяких компонентів може призвести до того, що зміняться характеристики параметрів захищеннності;

3. мінімальна кількість навчальних прикладів має бути в 10-20 разів більшою за кількість вхідних параметрів за формулою 2:

$$P_{\min} \geq (10...2o)N_X \tag{2.2}$$

де $P_{\min}$ – мінімальна кількість навчальних прикладів,

N_X – кількість вхідних параметрів НМ;

4. число навчальних прикладів повинно бути обмеженим;

5. варіанти навчальної вибірки мають показувати усі класи пропорційно, для розпізнавання їх НМ.

Згідно з цим створення необхідної кількості навчальних прикладів може сприяти виникненню проблем, які мають відношення до масштабування навчальних даних (вимоги 1, 2). Так само може відбуватись зі знаходженням необхідного обсягу інформації на основі статистики (вимоги 3, 4, 5). Необхідно уточнити, що дана проблема відноситься до методу виявлення аномалій та йде від формування ШНП. У цьому випадку для IDS має місце проблема знаходження статистичної інформації.

Найбільшу за часом тривалість створення НМ можемо знайти за формулою 3:

$$T_f \leq T_a \,, \tag{2.3}$$

де T_f – максимально допустима тривалість розробки НМ,

T_a – час, за який ризик від створення атаки не перебільшує вказану межу.

Для знаходження T_a можна скористатися методами експертних оцінок.

Якщо брати до уваги формулу 4

$$T_f = T_{\max} + t, \tag{2.4}$$

тоді маємо формулу 5

$$T_{\max} = T_f - t, \tag{2.5}$$

де $T_{\max}$ – максимальний час, за який формується навчальна вибірка,

t –період навчання НМ.

Період навчання НМ сильно залежить від архітектури НМ.

Детектор має як знаходити аномалії, так і виявляти атаки. Окрім цього, необхідно, щоб IDS час від часу проводило активне скануванням для знаходження слабких місць.

Через необхідну величину роботи для створення нормального робочого детектора у даному проекті розглядається та вирішується задача побудови концепту модуля виявлення аномалій. Будемо вважати, що дані, які поставлені модулем декодування, знаходяться у зрозумілій формі, тобто як набір певних ознак, які подіються як вхідні дані мережі.

Детектори аномалій працюють за приципом порівняння поточних даних з еталонними, та повідомлення, якщо вони перейшли допустиму межу.

Кожна модель має включати у себе такий еталон. Під час роботити НМ, еталон створюється під час навчання та правильної роботи мережі передачі даних (МПД), а згодом і мережу, яка під час функціонування знаходить похибки, які переходять вказані межі.

Недоліки та проблеми даного методу описані нижче:

– мережа повинна навчатися під час нормальної роботи МПД, проте неможливо точно дізнатися, чи не відбувається в цей момент атака, щоб мережа не вчилася на "аномальній активності";

– закінчити МПД або вимкнути її від мережі не можна, бо далі активність МПД може бути трактована, як аномальна;

– кількість даних та час навчання мають бути великих обсягів, бо результати роботи МПД різні у різний період доби та пори року.

Питання габаритної кількості даних можна зробити легшим, якщо використовувати адаптивні налаштування. Через це пропонується користуватися кластеризацією "картини мережевої активності".

Загалом задача кластеризації у нейромережах вирішується з використанням систем, які самоорганізовуються картиами Кохонена.

Додатковими проблемами, які вносить даний підхід, є:

Проте даний метод має ряд недоліків:

– вимоги вказання кількості кластерів;

– вимоги перенавчання детектора, якщо змінювати топологію МПД.

Для вирішення перелічених проблем, для створення детектора аномалій було прийнято рішення скористатися нейромережевими алгоритмами кластеризації.

Ця ідея поглягає у тому, що мережа навчається у нормальному режимі у рамках певного періоду. Під час даного часу створюється певна кількість

кластерів даних. Вони при коректних налаштуваннях мережі, у повному обсязі ваідображають нормальну роботу МПД.

Існують вказані варіанти виявлення:

– навчання мережі на поступаючих даних. При виникненні аномалій, структура і кількість кластерів змінюються;

– надання даних, що потрапили до навченої мережі і перевірка того, як приблизно вона відповідає певному кластеру;

– У першому випадку, аномалії можна побачити при:

– зміни кількості кластерів;

– динамічній зміні швидкості появи та зникнення нових нейронів і зв'язків.

У другому випадку аномалії можна побачити за зміною від норми показників нових вимірів на відміну від середнього знаячення для даних, по яким мережа вчилася.

У мережевих алгоритмах розрахунок динаміки виконати складно: мережа має змогу змінювати кількість нейронів і зв'язків під час кожної з ітерацій, при умові, що певні алгоритми створюють та знищують певну кількість нейронів під час кожної ітерації.

Зміна кількості кластерів не вважається надійним показником. Також варто зауважити, що навчання мережі породжує багато різних проблем (наприклад, перенавчання) Проте вимір відхилень є порівняно надійним методом. Тому саме цей метод треба використовувати [10].

2.2 Архітектура модуля

Показники з пристроїв відправляються агентами і для будь-якого критерію вираховується середнє. Дані показники отримуються від IDS, групуючись в спільний вектор з інформацією активності мережі. Мережеві показники подаються пристрою IDS у типовій формі. Після цього для кластера вираховується середнє арифметичне усіх величин відхилень потрапляючих нейронів.

Для зразку віднайдемо найближній нейрон прикладу даних, шукається проміжок між ними та зіставляємо з порахованою середньою квадратичною розбіжністю для кластера, що містить нейрон.

Приклад є аномальним за умови, коли проміжок між ним та нейроном є більшим за відхилення по кластеру.

Це є гібридом прийому класифікації по середньому інтервалу, що активно експлуатується для встановлення розряду нейронів та способу поділу по найменшому інтервалу.

Спершу застосовуємо перший метод – обираємо зразковий нейрон, а потім рахуємо відмінність між середньою розбіжністю по кластеру.

Існує ще один спосіб – це розподіл методом більшості, тобто той процес, коли нейрон приймає клас по найбільшому числу точок в своїй області.

Основа детектора є елементом, що здійснює один із механізмів "зростаючого нейронного газу" або GNG, що пристосований на активність мережі у процесі нормальної діяльності МПД.

GNG має змогу кластеризувати показники, при цьому особисто вказувати число кластерів, що потребується.

Наприклад, це можливо один із даних механізмів:

– GNG - класичний зростаючий нейронний газ

– IGNG - інкрементальний зростаючий нейронний газ.

Пристрій залишає дані, завдяки яким він отримав знання, як мережеву структуру й приймає нові матеріали в ході робочого процесу.

Модуль має увесь час показувати підсумок на випуску: чи існує аномальна діяльність, чи її немає, або чи можливе існування цієї активності.

2.3 Зростаючий нейронний газ

Для реалізації був обраний метод зростаючого нейронного газу. GNG - це алгоритм, який має змогу створювати пристосувальну кластеризацію вхідної інформації, тобто не лише поділити зону на кластери, а також з’ясувати необхідне їхнє число на основі індивідуальностей цих даних.

Алгоритм по порядку трансформує кількість нейронів (частіше збільшує, розпочинаючи з двох елементів), у цей же час складаючи комплект зв’язків між нейронами (найкращим способом підходить поділу вхідних векторів), при цьому застосовуючи метод загального навчання Хебба.

У будь-якого нейрона є внутрішній параметр, в якому збирається т.зв. «локальна помилка». Зв'язок між пристроями кваліфікуються змінною, яка має назву «вік».

Алгоритм GNG може бути представлений у такому вигляді:

1. ініціалізація: формування двох вузлів за участі векторів ваг, які дозволяються поділом початкових векторів, і змістом локальних помилок, що дорівнює нулю; зробити вік ріним нулю, об’єднавши вузли з’єднаннями;
2. дати на вхід мережі нейронів вектор X;

3. виявити два нейрона s і t, найближніх до x, тобто пристрої з векторами ваг w_s і w_t, такими, що $\|w_s - x\|^2$ - найменше, а $\|w_t - x\|^2$ - другий найменшого значення проміжку з-поміж усіх пристроїв;

4. відновити локальну помилку нейрона-переможця s методом доповнення до неї відстані у квадраті між векторами w_s і x:

$$E_s \leftarrow E_s + \|w_s - x\|^2 \tag{2.5}$$

5. відсторонити нейрон-переможця s та його всіх системних межуючих нейронів в бік вхідного вектора x на проміжок, таких же часток ϵ_w і ϵ_n від усього;

$$w_s \leftarrow w_s + \epsilon_w \cdot (w_s - x) \tag{2.6}$$

6. продовжити на одиницю вік усіх зв'язків, які йдуть від переможця s;

7. коли два найкращих нейрона s і t пов’язані, то тоді маємо вік їхнього контакту прирівняти до нуля. В іншій ситуації маємо побудувати контакт між нейронами;

8. виключити усі зв’язки, вік яких є більшим за $age_{\max}$. Якщо згодом з’являються нейрони, що не володіють контактами з іншими пристроями, усунути дані нейрони;

9. якщо номер даного повторення є кратним λ, і рубіжний мережевий розмір не створено, скласти новий нейрон r з даними умовами:

– відшукати нейрон u з максимальною локальної помилкою;

– посеред межуючих нейронів u відшукати нейрон v з найбільшою помилкою;

– створити пристрій r "Посередині" між u і v:

$$w_r = \frac{w_u + w_v}{2} \tag{2.7}$$

– перемінити зв'язок між u і v на зв'язку між u і r, v і r;

– зробити похибку нейронів u і v меншою, покласти показник помилки нейрона r;

10. Зробити похибки всіх нейронів меншою на j частку β.

$$E_j \leftarrow E_j - E_j \cdot \beta \tag{2.7}$$

11. При умові не зупинки критерію треба повернутися на крок 2.

Розглянемо процес привчання зростаючого нейронного газу до деяких властивостей вхідних просторів.

По – перше, необхідно роглянути зростання перемінної лоепльної похибки у найкращого нейрона на 4 етапі. Дані дії призводять до такої ситуації, коли переможцями частіше стають вузли , в окіл яких попадають багато вхідних сигналів та мають найбільшу помилку. Тобто можна зробити висновок, що ці області і є тими самимим претендентами на зменшення. Це відбувається додаючи інші вузли.

Під час проходження 5 єтапу може відбуватися зміна місцезнаходження вузлів ближче до вхідного вектора. Це є ознакою того, що нейрон, який переміг, хоче «усереднити» свій стан у області вхідних сигналів, які

знаходяться у його околі. У такій ситуації варто зауважити, що переможений нейрон зсувається у сторону, звідки поступає сигнал, та тягнеться до сусідніх нейронів. (зазвичай, обирається $\epsilon_w \sim 100...1000\epsilon_n$).

Тепер необхідно трохи розповісти про нейронні зв'язки та їх взаємодію на 6 – 8 етапах. Уся технологія старіння та викидання вже відпрацьованих зв'язків будується на найбільшого зближення топології мережі та тріангуляції Делоне.

Іншими словами, тріангуляція Делоне показує найкрасивіше масимальне значення ентропії топологізація шару. Варто відмітити, що структура топології та її значення є важливим лише під час знаходження нового місця для вузлів, якщо додпавати їх на 8 етапі. Також слід сказати. Що вони завжди розташовуються посередині ребер.

На 9 етапі відбувається покращення похибок усіх нейронів певного шару. Ця процедура проводится для викреслення з пам'яті старих вхідних векторів для ефективнішої реакції на нові. Тобто з'являється такі варіанти використання нейронного газу для пристосування НМ під повільні розподіли вхідних сигналів. Але це не допомагає вираховувати та бачити зміни показниуів входу.

Розглянемо критерії зупинки. У цьому випадку алгоритм покладається на роботу розробників аналітичних систем. Серед існуючих можливостей – випробування мережевої ефективності на множині тестування, аналітика зміни середнього значення похибки нейронів, обмеження складності роботи мережі і т.д.

2.4 Висновки

В цьому розділі була розглянута вибраний принцип роботи детектора аномальної активності та архітектура модуля по виявленню активності, що базується на ньому. Була розглянута вся необхідна теорія, що дозволяє приступити до створення програмної реалізації.

3 ОПИС СТВОРЕНОЇ ПРОГРАМИ ДЛЯ РЕАЛІЗАЦІЇ ДАНОГО РІШЕННЯ

3.1 Опис програмних засобів для розробки рішення

Програмний модуль реалізований за допомого мови програмування Python 3.6. Середовище розробки, що було використано для реалізації – Jupyter Notebook. Необхідне програмне забезпечення для роботи програми:

– Python 3.6;

– Jupyter Notebook.

Додаткові програмні модулі:

– Scipy;

– Sklearn;

– Numpy;

– Os;

– Time;

– Networkx.

Короткий опис додаткових програмних пакетів.

Scipy - бібліотека для Python з відкритим кодом, призначена для виконання різного роду математичних операцій. Включає в себе налаштування, що допомагають при оптимізації, створенні реалізацій різного роду генетичних алгоритмів, обробки сигналів та інших математичних задач.

Sklearn – безкоштовна бібліотека, призначена в основному для машиного навчання на мові Python. Має вбудовані алгоритми класифікації, кластеризації, навчання мережі різного ступеня складності.

Numpy - це розширення мови Python, що додає підтримку великих багатовимірних масивів і матриць, разом з великою бібліотекою

високорівневих математичних функцій для операцій з цими масивами. Розглядається як альтернатива MATLAB, дає можливість реалізації математичних алгоритмів з більшою швидкістю. Numpy має широкий функціонал для роботи та виконання математичних операцій над багатовимірними масивами.

Time – модуль, необхідний для роботи з часом. В програмі використовувався для замірів часу роботи програми, часу навчання моделі, часу, витраченого на обробку записів.

OS - модуль, необхідний для роботи з операційною системою. Модуль є цілком кросплатформений, тому програмний застосунок залишається цілком сумісний з усіма системами, що виконую код на Python.

3.2 Опис вхідних даних

Програма являє собою модуль по виявленню аномальної активності трафіку комп'ютерної системи. В якості вхідних даних була використана база NSL-KDD. Це модифікована база даних KDD-99, що має ряд переваг в порівнянні з оригінальною:

– не має повторюваних записів;

– рівномірне розподілення видів записів, що дозволяє отримувати більш точні результати від класифікаторів;

– оптимальний розмір вибірок, що дає змогу проводити навчання без додаткового вибору окремих підвибірок.

Вхідний датасет є базою даних активності трафіку певної комп'ютерної системи. Для навчання з цілого набору даних відібрано 1011 записів. Після

навчання моделі програма аналізує спочатку весь тренувальний набір даних, далі програма виконується для всієї вибірки.

Набір NSL-KDD має багато зібраних даних. Для аналізу було відібрано лише деякі параметри.

Список параметрів, відібраних до аналізу:

– тривалість з’єднання;

– тип протоколу;

– мережева служба отримувача;

– поточний стан з’єднання;

– кількість байт, що передана отримувачу;

– кількість байт, яку відправив отримувач;

– кількість помилкових пакетів;

– кількість пакетів з поміткою URG;

– кількість підключень до вузла за останні дві секунди;

– кількість підключень до цього сервера за останні дві секунди;

– % підключень зі спробою встановити підключення з помилкою;

– % підключень до інших серверів;

– % підключень до інших вузлів.

Ці дані було взято, зважаючи на потреби в результаті нашої програми. При комерційній розробці треба більш детально вивчити параметри даних, що збираються датчиками, параметри протоколів, потреби конкретної реалізації і підбирати параметри під конкретні ситуаційні потреби. Розроблювана програма є цілком універсальною, дозволяє додавати або прибирати з переліку параметрів, що використовуються для попередження загроз певний параметр. Це є вагомим плюсом реалізації даної нейрнонної мережі та модуля виявлення загроз безпеці, адже даний програмний застосунок можна застосовувати для

аналізу захищеності комп’ютерних мереж різних типів та конфігурацій. Потрібно лише вибрати такі параметри для аналізу, що будуть актуальні в тій чи іншій конкретній комп’ютерній системі.

3.3 Опис реалізації програми

Ціль роботи програми – з вхідного набору даних класифікувати дані на аномальні та нормальні, попередньо навчивши програмний модуль нейронної мережі розпізнавати аномальні та нормальні дані. Схему роботи програми представлено на рисунку 1.

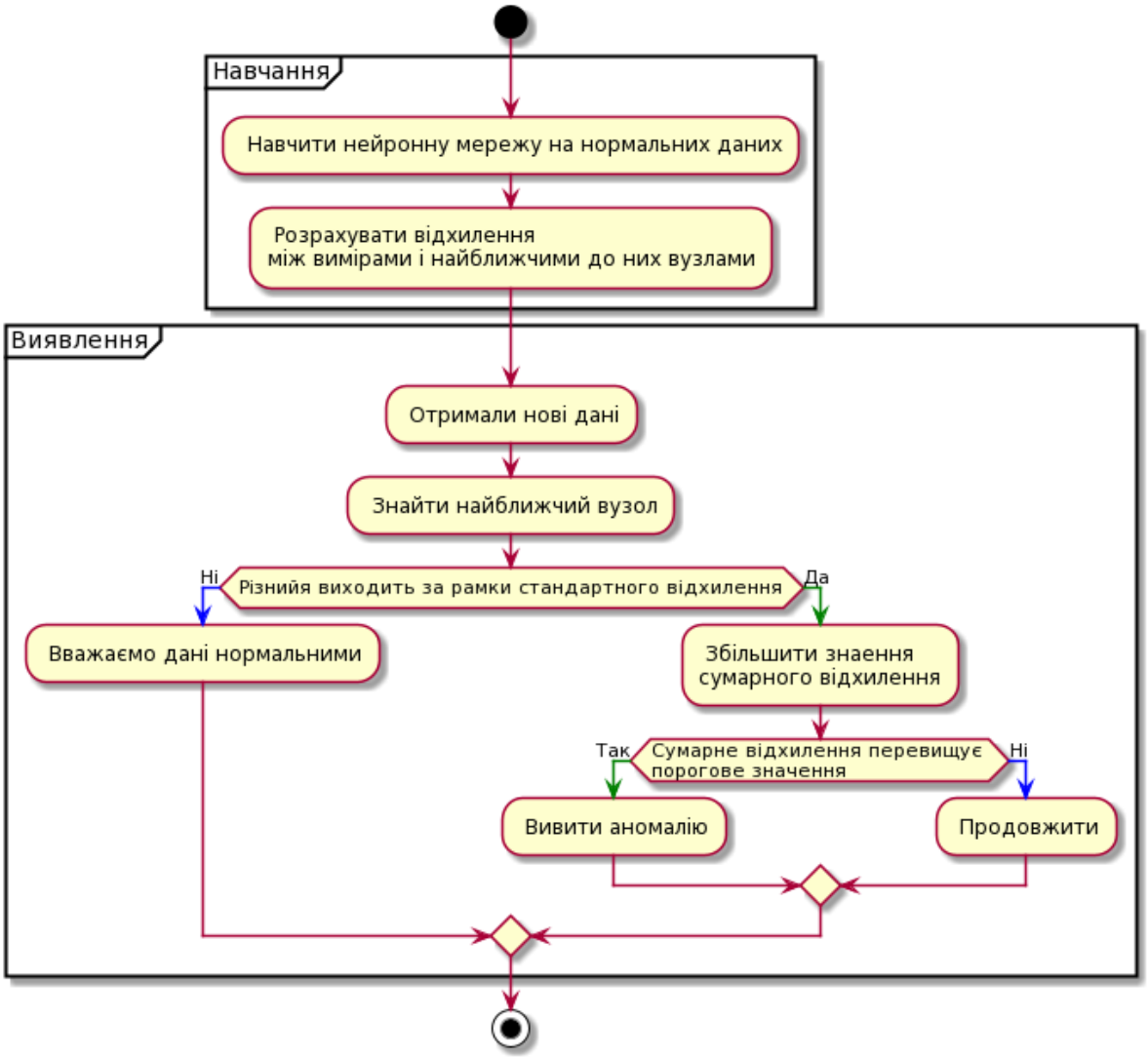

Рисунок 3.1 – Схема роботи програми

На рисунку 2 наглядно видно, що програмний продукт складається з двох модулів, що можуть працювати незалежно: модуль нейронної мережі та модуль детектора загроз.

За основу нейронної мережі була взята реалізація алгоритму нейронного газу, що знаходиться у відкритому доступі. Ця реалізація була адаптована під потреби програмного застосунку. Зокрема, реалізація алгоритму була

перероблена під багатовимірний випадок, як цього вимагає вхідний масив з даними.

Для роботи програми створено файл з розширенням .csv з вибіркою, необхідною для навчання нейронної системи. Також файл з усією вибіркою було переформатовано на формат .csv.

Алгоритм роботи програми.

Програма написана в об’єктно-орієнтовному стилі. Починає свою роботу зі стандартного методу main(). В цьому методі відбувається налаштування всіх необхідних параметрів, необхідних для роботи детектора аномальної активності мережі. Після цього метод main() запускає виконання методу detection() з відповідними параметрами, який виконує далі всі необхідні операції для пошуку аномальної активності.

Метод detection() завантажує всі необхідні дані для навчання нейронної мережі. З отриманими даними в якості параметра далі викликається метод train(), що і проводить навчання нейронної мережі. Метод train() в якості параметрів приймає також максимальну кількість ітерацій, яку буде працювати навчання мережі. Ще одним необов’язковим параметром цього методу є також кількість ітерацій, після якої зберігається візуалізація побудованої нейронної мережі. Після навчання нейронної мережі виконується метод detect_activity(), який і виявляє аномальну активність трафіку. Метод визивається декілька разів: для вибірки, на якій набір навчений, для цілого набору даних, з якого сформована навчальна вибірка, та для всього набору даних.

Методи, де необхідно зчитувати дані з файлів, виконують функцію read_data(). Вона інкапсулює в собі стандартний модуль Python – csv. Булевий параметр activity, що може приймати 2 значення, задає тип даних, що знаходяться у файлі – нормальні або з підозрілою активністю. Також

реалізована функція hostactivity(), що за вимогою генерує дані вузла. За те, чи будуть генеруватися дані про активний трафік з вузла, відповідає параметр host_data функції detect_activity(), що може приймати значення 0 або 1. Результатом роботи функції hostactivity() є ненормалізований масив Numpy. Всі дані, як зчитані з файлу так і опціонально згенеровані дані про активність вузла передаються до конструктору головного класу детектора аномальної активності. В середині екземпляру класу, створеного конструктором, всі дані запам’ятовуються у внутрішні властивості класу. Головним класом, агрегованим в детектор є клас з назвою GNG(*англ.Growing Neural Gas).* В цьому класі реалізовані базові методи для моделювання і тренування нейронної мережі, функцію активації нейрона, функції додавання нових нейронів та нових зв’язків.

3.4 Висновки

В даному розділі було розглянуто особливості створення та принцип роботи конкретної реалізації по виявленню аномальної активності. Був створенний опис всіх методів, наведені необхідні умови, що стосуються програмних вимог для можливості функціонування програмного продукту. Тепер ми маємо змогу проаналізувати результати, що видає програма.

4 АНАЛІЗ РЕЗУЛЬТАТІВ ЕКСПЕРИМЕНТІВ

4.1 Початкові умови

Для навчання програми використовувався датасет з 1000 рядків. При цьому, при навчанні відбираліся лиши дані з нормальною активнісю, яких було 516. Навчання відбувалося на вибірці в 516 елементів, умовою закінченння було проведення 7000 ітерацій. Час навчання, затрачений мережею на обучення склав 6800 секунд , що складає майже дві години. Цей час не є непомірно великим, адже вибірка, що була взята за основу для навчання мережі, була об’ємом тисяча символів. Але й отриманий час навчання можна було прискорити, використавши для навчання хмарні рішення, такі як TensorFlow, Google Cloud Ai, Kaggle. Ці рішення є безкоштовними, і дозволяють використовувати свої потужності для некомерційного навчання власних нейронних мереж та моделей. Але, оскільки час роботи не був занадто високим, було вирішено не використовувати сторонні хмарні рішення, а цілком відтворювати весь цикл програмного продукту на власному персональному комп’ютері.

4.2 Аналіз результатів експериментів з різними вхідними даними

Нейронна мережа розбила вхідний масив даних на 78 кластерів. Після закінчення навчання мережі програму запустили спочатку для нормальних даних з тренувального датасету. Результати роботи представлені на рисунку 1.

```
----------------------------------------------------------------------
Abnormal records = 0, Normal records = 1, Detection time = 0.34 s, Time per record = 0 s
Abnormal records = 0, Normal records = 101, Detection time = 0.41 s, Time per record = 0.00411898136138916 s
Abnormal records = 0, Normal records = 201, Detection time = 0.47 s, Time per record = 0.0023588716983795168 s
Abnormal records = 0, Normal records = 301, Detection time = 0.53 s, Time per record = 0.0017652837435404459 s
Abnormal records = 0, Normal records = 401, Detection time = 0.6 s, Time per record = 0.0014960002899169923 s
Abnormal records = 0, Normal records = 501, Detection time = 0.65 s, Time per record = 0.0012965316772460937 s
Anomalies weren't detected [abnormal records = 0, normal records = 516, detection time = 0.66 s, time per record = 0.001273729080377623 s]
----------------------------------------------------------------------
```

Рисунок 4.1 – Результати роботи нейронної мережі

Як видно, для набору з тренувальних даних програма працює зі 100% точністю, що і не дивно, адже програме зберігає дані про тренувальний датасет у вигляді нейронної мережі. Далі програму було запущено для записів з тренувальної вибірки з аномальною активністю. Результати роботи програми на цій вибірці представлені на рисунку 2.

```
Reading abnormal activity from the file "NSL_KDD/Small Training Set.csv" [generated host data was not included]...
Records count: 495
Abnormal records = 0, Normal records = 1, Detection time = 0.0 s, Time per record = 0 s
Abnormal records = 65, Normal records = 4, Detection time = 0.13 s, Time per record = 0.0012865543365478515 s
Abnormal records = 140, Normal records = 8, Detection time = 0.18 s, Time per record = 0.0008826351165771484 s
Abnormal records = 212, Normal records = 15, Detection time = 0.22 s, Time per record = 0.0007464853922526042 s
Abnormal records = 287, Normal records = 21, Detection time = 0.27 s, Time per record = 0.0006769669055938721 s
Anomalies were detected (count = 7) [abnormal records = 465, normal records = 30, detection time = 0.32 s, time per record = 0.0006417394888521445 s]
----------------------------------------------------------------------
```

Рисунок 4.2 - Результати роботи програми на вибірці

Ми можемо спостерігати незначне зменшення точності зі 100% до 93 відсотків. Окремий аналіз загроз на аномальній та нормальній активності потрібен для того, щоб визначати процент помилки нейронної мережі. Далі програма була запущена для всієї тренувальної вибірки даних. Результат роботи програми представлено на рисунку 3.

```
Records count: 1011
Abnormal records = 0, Normal records = 1, Detection time = 0.0 s, Time per record = 0 s
Abnormal records = 31, Normal records = 70, Detection time = 0.05 s, Time per record = 0.0004787445068359375 s
Abnormal records = 61, Normal records = 140, Detection time = 0.09 s, Time per record = 0.0004737448692321777 3 s
Abnormal records = 92, Normal records = 209, Detection time = 0.15 s, Time per record = 0.0004835279782613184 s
Abnormal records = 130, Normal records = 271, Detection time = 0.2 s, Time per record = 0.0004946929216384888 s
Abnormal records = 165, Normal records = 336, Detection time = 0.25 s, Time per record = 0.0004935660362243653 s
Abnormal records = 201, Normal records = 400, Detection time = 0.3 s, Time per record = 0.000497744878133138 s
Abnormal records = 234, Normal records = 467, Detection time = 0.36 s, Time per record = 0.0005092712811061314 s
Abnormal records = 276, Normal records = 495, Detection time = 0.41 s, Time per record = 0.0005128863453865051 s
Abnormal records = 313, Normal records = 508, Detection time = 0.47 s, Time per record = 0.000520212385389 54 s
Abnormal records = 356, Normal records = 545, Detection time = 0.52 s, Time per record = 0.0005160577297210694 s
Anomalies were detected (count = 7) [abnormal records = 465, normal records = 546, detection time = 0.52 s, time per record = 0.0005153758353696496 s]
---------------------------------------------------------------------
```

Рисунок 4.3 - Результат роботи програми для всієї тренувальної вибірки даних

Як видно, результатом останнього прогону програми на всій тренувальній вибірці є сума двох попередніх результатів. Після закінчення експериментів з тренувальною вибіркою програму було запущено для всього набору даних. Знову виявлення загроз безпеці коп’ютерної системи було поділено на 2 етапи. Спочатку програма намагалася виявити загрози безпеці в даних з підозрілою активністю. Результат роботи програми для цієї вибірки представлений на рисунку 4.

```
Reading abnormal activity from the file "NSL_KDD/KDDTest-21.txt" [generated host data was not included]...
Records count: 9698
Abnormal records = 1, Normal records = 0, Detection time = 0.0 s, Time per record = 0 s
Abnormal records = 783, Normal records = 218, Detection time = 0.49 s, Time per record = 0.0004850766658782959 s
Abnormal records = 1564, Normal records = 437, Detection time = 1.04 s, Time per record = 0.0005210708379745484 s
Abnormal records = 2348, Normal records = 653, Detection time = 1.54 s, Time per record = 0.0005148481527964275 s
Abnormal records = 3141, Normal records = 760, Detection time = 2.12 s, Time per record = 0.0005293816328048706 s
Abnormal records = 4932, Normal records = 869, Detection time = 2.61 s, Time per record = 0.0005223369598388672 s
Abnormal records = 5739, Normal records = 902, Detection time = 3.18 s, Time per record = 0.0005294680595397949 s
Abnormal records = 6561, Normal records = 914, Detection time = 3.66 s, Time per record = 0.0005225808961050851 s
Abnormal records = 7344, Normal records = 995, Detection time = 4.24 s, Time per record = 0.0005297485888004303 s
Abnormal records = 8131, Normal records = 1007, Detection time = 4.72 s, Time per record = 0.0005245452457004123 s
Anomalies were detected (count = 154) [abnormal records = 8671, normal records = 1027, detection time = 5.15 s, time per record = 0.0005306979262182259 s]
---------------------------------------------------------------------
```

Рисунок 4.4 - Результат роботи програми для вибірки

Як видно з рисунку, процент помилки склав 10%, відповідно точність впала до 90%. Результат погіршився порівняно з попереднім експериментом, точність впала на 3 відсотки, але і розмір вибірки для аналізу виріс в 9 разів, що з більшою дисперсією впливає на точність даного аналізатора підозрілої активності.

Після цього програма була запущена для аналізу заздалегідь нормальних даних з всієї вибірки. Результат роботи програми для цієї вибірки даних представлено на рисунку 5.

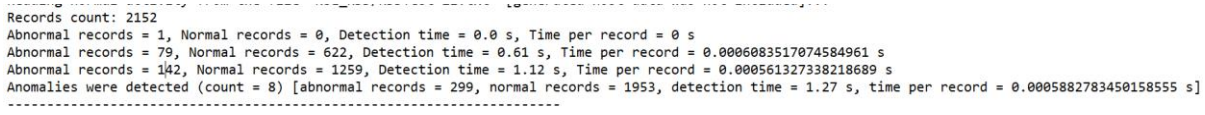

Рисунок 4.5 - Результат роботи програми для вибірки даних

Як бачимо, програма помилилася в 13% випадках, прийнявши нормальні дані за дані з підозрілою активністю і сигналізувавши про загрозу. Фінальний запуск аналізатора загроз комп'ютерних систем було проведено на повній вибірці даних. Результат представлений на рисунку 4.6.

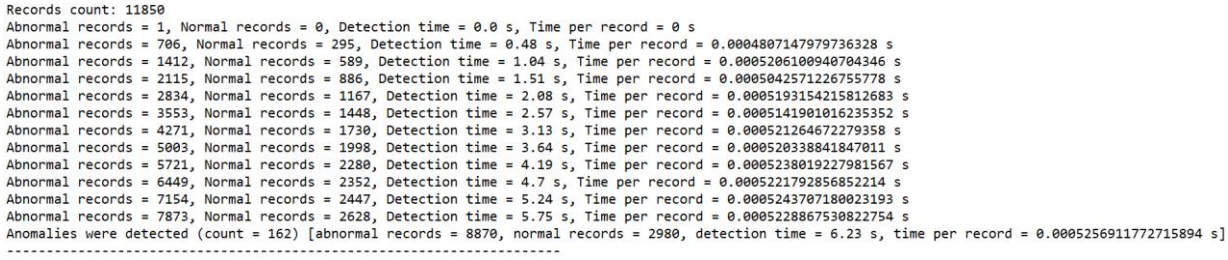

Рисунок 4.6 – Результат фінального запуску аналізатора

Зважаючі на отримані результати програми, можна зробити висновок, що приблизний процент помилки – 11.5%, відповідно точність, з якою програма класифікує трафік на нормальний та з підозрілою активністю складає майже 90%.

Перспективи покращення алгоритму:

– додати адаптивне навчання;

– для поліпшення виявлення, треба враховувати порядок проходження подій. Лише частково він міг би враховуватися з використанням адаптивного підлаштовування, яке зараз не працює;

– можливо здійснювати зупинку GNG за індексом Калінського-Харабаза;

– існує більш ефективний алгоритм - Fast GNG (приблизно в 50 разів швидше, ніж GNG), також існують GPU реалізації GNG. Безсумнівно, що цей факт можливо використовувати для прискорення;

4.3 Висновки

В цьому розділі був проведений аналіз експериментів з різними типами вхідних даних. Нейромережевий метод виявлення аномальної активності досить непагано впорався з усіма типами вхідних даних. Також було наведено перспективи покращення даної реалізації.

5 ФУНКЦІОНАЛЬНО-ВАРТІСТНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

5.1 Постановка задачі

Проводиться оцінка основних характеристик отриманих результатів від кінцевого програмного продукту та його собівартість. Для отримання практичних результатів була використана мова програмування Python 3. Середовище розробки - PyCharm. Нижче наведені різні підходи до отримання потрібних результатів від нейронної мережі та їх аналіз з метою побудови оптимальної реалізації з технічної та економічної точки зору.

5.2 Обгрунтування функцій дослідження

Основні функції:

F_1 – об’єм вхідного датасету з даними; F_2 – вибір алгоритму кластеризації даних;F_3 – вибір мови програмування для реалізації програмного продукту;

Функція F_1 – а) 5000 ситуацій; б)11000 ситуацій; в) 200000 ситуацій;

Функція F_2 – а) нейронний газ; б); нейронна мережа прямого поширення

Функція F_3 – а) Мова розробки Python 3; б) Мова розробки JavaScript; в) Мова розробки C++;

Знизу зобразимо відповідну морфологічну карту системи на рисунку 5.1:

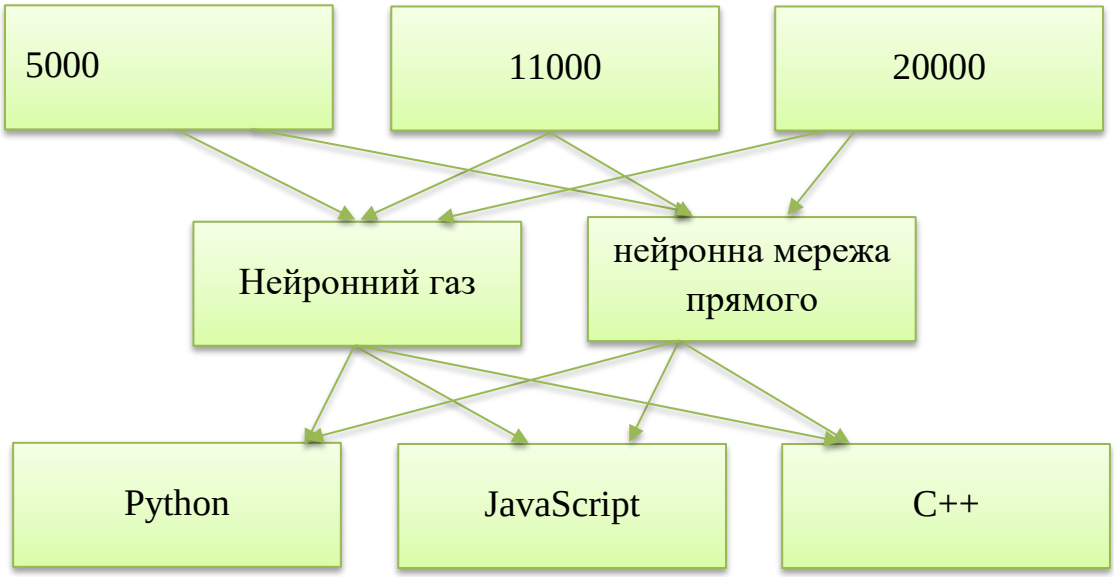

Рисунок 5.1 - Морфологічна карта системи

Порівняльна таблиця представлена у таблиці 5.1

Таблиця 5.1 – Порівняльна таблиця

Функції	Варіанти реалізації	Переваги	Недоліки
F_1	А	Велика швидкість обробки даних і отримання результатів нейронною мережею	Недостатньо даних для коректного навчання нейромережі

F_1	Б	Більша відповідність об’єму даних та коректності навчання, прийнятна швидкість роботи програмного продукту	Більший об’єм даних займає більше місця
F_1	В	Найбільш коректне навчання нейронної мережі	Занизька швидкість роботи програмного продукту, займає багато місця
F_2	А	Адекватний час навчання, висока ефективність та результативність мережі	Досить складна для розуміння та побудови
F_2	Б	Проста теорія для розуміння та побудова мережі	Потребує багато часу для навчання та ресурсів комп’ютеру, а також часу для

			налаштувань мережі
F_3	А	Велика кількість готових та протестованих бібліотек для реалізації наших завдань, достатня кількість літератури для вивчення	Менша швидкість роботи
F_3	Б	Більші можливості для візуалізації результатів	Менша варіативність у моделюванні, менша швидкість роботи
F_3	В	Висока швидкість роботи	Менші можливості для реалізації, мало готового функціоналу для моделювання

З позитивно-негативної матриці можна робити висновки щодо доцільності використання певних варіантів та недоцільність інших:

Після порівняльного аналізу варіантів реалізації основних функцій по їх перевагам та недолікам можна виключити наступні варіанти F_1 А і Б , F_3 Б і В, тоді варіанти, які залишилися:

F_1 В) $->$ F_2 А -> F_3 А

F_2 Б) $->$ F_2 А -> F_3 А

Для наглядної оцінки запропонованих функцій введемо наступну систему параметрів.

5.3 Обґрунтування системи параметрів досліджень

Для характеризації досліджень запропонуємо такі параметри:

X1 – час навчання нейронної мережі (у годинах); X2 – час освоєння теоретичної бази нейронної мережі(у годинах); X3 – необхідний час для вивчення мови програмування (у годинах);X4 – використання ресурсів процесора (у відсотках).

Функція F3 залежить від двох параметрів – X3 та X4. Дані параметрів представлені у таблиці 2.

Таблиця 5.2 - Дані параметрів X3 та X4

Умовні позначення	Одиниці виміру	Значення параметрів		
		Гірші	Середні	Кращі

X1	Год	4	2	1
X2	Год	20	15	10
X3	Год	40	20	15
X4	%	30	15	5

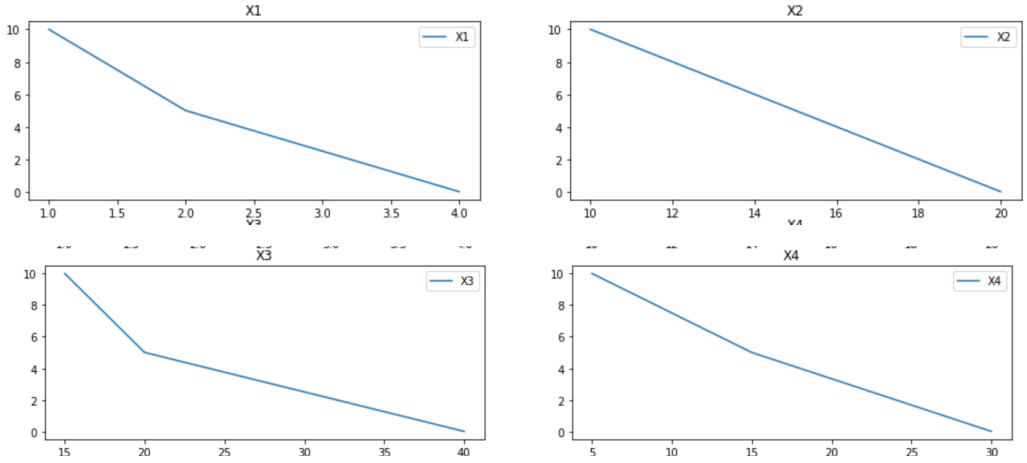

Рисунок 5.2 – графіки оцінок

Вагомість параметрів вираховується за допомогою методу попарного їх зрівняння на основі результатів ранжування експертами та порівняння параметрів попарно представлені у таблиці 3.

Таблиця 5.3 - Вагомість параметрів

Параметр	Ранг параметру по оцінці експерта							Сума рангів, R_i	ВідхиленняΔ_i	Квадрат відхилення, $(\Delta_i)^2$
	1	2	3	4	5	6	7			
X1	1	2	1	2	1	1	1	9	-8.5	72,25
X2	3	3	2	1	2	2	2	15	-2.5	6,25
X3	2	1	4	4	3	4	3	21	3,5	12,25
X4	4	4	3	3	4	3	4	25	7,5	56,25
Разом	10	10	10	10	10	10	10	70	0	171

Будемо вважати 1 ранг найнижчим а 4 – найвищим. Тому експерт поставить найважливішому, на його думку, параметру 4, а найменш важливому відповідно 1.

Вирахуємо коефіцієнт конкординації та представимо у формулі (1).

$$W = \frac{12S}{N^2(n^3-n)} = \frac{12*171}{7^2(4^3-4)} = 0.71 > W_k = 0.67, \qquad (5.1)$$

де

Оскільки коефіцієнт конкординації більше нормативного то результати вважаємо достовірними.

Таблиця 5. 4 – Попарне зрівняння параметрів

Параметри	**Експерти**							**Кінцева оцінка**	**Числове значення**
	1	**2**	**3**	**4**	**5**	**6**	**7**		
X1 та X2	<	<	<	>	<	<	<	<	0,5
X1 та X3	<	>	<	<	<	<	<	<	0,5
X1 та X4	<	<	<	<	<	<	<	<	0,5
X2 та X3	<	<	<	<	<	<	<	<	0,5
X2 та X4	<	<	<	<	<	<	<	<	0,5
X3 та X4	<	<	>	>	<	>	<	<	0,5

Таблиця 5.5 - Розрахунок вагомості параметрів

Параметри	Параметри				Перший крок		Другий крок		Третій крок	
	X1	X2	X3	X4	bi	Kbi	bi’	Kbi’	bi’’	Kbi’’
X1	1	0,5	0,5	0,5	3,5	0,175	11,75	0,156	43,375	0,1577
X2	1,5	1	0,5	0,5	4,5	0,225	15,75	0.21	57,125	0,207
X3	1,5	1,5	1	0,5	5,5	0,275	20,75	0.276	75,375	0,274
X4	1,5	1,5	1,5	1	6,5	0,325	26,75	0.356	99,125	0,360

Таблиця 5.6 – Розрахунок рівня якості варіантів реалізації

Основна функція	Варіант реалізації	Параметри	Абсолютне значення параметру	Бальна оцінка параметру	Коефіцієнт вагомості параметру	Коефіцієнт якості
F_1	Б	X1	1.4	8	0,1577	1,2616
F_1	В	X1	3.0	3	0,1577	0,55
F_2	А	X2	16	4	0,207	0.828
F_3	А	X3	19	5.8	0,274	1.58
F_3	А	X4	6	9,2	0,360	3,312

K1 = 1.2616+0.828+1.58+3.312 = 6.9816

K2 = 0.55+0.828+1.58+3.312 = 6.27

Варіант з вибіркою у 11000 даних на виході має вищий коефіцієнт технічного рівня, отже є кращим

5.4 Економічний аналіз варіантів розробки ПП

Кожен з варіантів реалізації включає в себе наступні етапи:

– Створення програмного продукту

– Пошук та обробка знайдених даних

1) Власноручний збір даних
2) Пошук готової БД

– Створення моделі та її навчання для конкретного програмного продукту

Для завдання 1 (Алгоритм складності 3, ступінь новизни Г, вид використаної інформації БД):

T_p =8, $K_{\text{п}}$ = 0,3, $К_{\text{ск}}$= 0,8, $K_{\text{ст.м}}$ = 1.3

T_1 = 8*0.3*0.8*1.3 = 2,496

Для завдання 2 при реалізації першого варіанту(Алгоритм складності 1, ступінь новизни Б, вид використаної інформації ПІ):

T_p =64, $K_{\text{п}}$ = 2,02, К_ст = 0,65, $K_{\text{ст.м}}$ = 1,35

T_{2_1} = 64 * 2,02 * 0,65 * 1,35 = 113,443 людино-днів

Для завдання 2 при реалізації другого варіанту (Алгоритм складності 1, ступінь новизни Б, вид використаної інформації ПІ) :

T_p =64, $K_{\text{п}}$ = 2,02 $К_{\text{ск}}$= 0,8, $K_{\text{ст.м}}$ = 1,55

T_{2_2} = 64* 2,02 * 0,8 * 1,55 = 160,307 людино-днів

Для завдання 3 (Алгоритм складності 1, ступінь новизни В, вид використаної інформації БД) T_p = 43, $K_{\text{п}}$ = 0,68, $К_{\text{ск}}$ = 0.7, $K_{\text{ст.м}}$ = 1.6

T_3 = 43 * 0,68 * 0,7 * 1,6 = 32,74 людино-днів

T1 = (2,496+ 113,443 + 32,74) * 8 = 1173,432 людиногодин

T2 = (2,496+ 160,307+ 32,74) * 8 = 1564,344 людиногодин

У процессі розробки програмного продукту приймають участь два програміста з місячним окладом 11 000 грн.

$$C = \frac{22\ 000}{2*14*9} = 87{,}3$$

Зарплата кожному варіанту:

$C_1 = 1173{,}432 * 87{,}3 = 102\ 440$ грн

$C_2 = 1564{,}344 * 87{,}3 = 136\ 567$ грн

На соціальний внесок відрахування становить 22%:

$C_{1v} = 0{,}22 * 102\ 440$ грн = 22536 грн

$C_{2v} = 0{,}22 * 136\ 567$ грн = 30044 грн

Визначимо витрати що потрібна на оплату однієї машино-години. Враховуючи що заробітна плата одного програміста складає 11 000 грн з коефіцієнтом зайнятості 0,215:

$C_г = 12 * М * K_з = 12 * 11000 * 0{,}215 = 28\ 380$ грн

Враховуючи додаткову заробітну плату:

$C_{зп} = C_г * (1 + K_з) = 28\ 380 * (1 + 0{,}215) = 34\ 481$ грн

Відрахування на соціальний внесок:

$C_{\text{від}} = C_{\text{зп}} * 0,22 = 7585,974$ грн

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10 000 грн

$C_{\text{А}} = 1,15 * 0,25 * 10000 = 2875$ грн

Підрахуємо витрати на профілактику та ремонт:

$$\text{С}_{\text{Р}} = \text{К}_{\text{ТМ}} * \text{Ц}_{\text{ПР}} * \text{К}_{\text{Р}} = 1,15 * 10000 * 0,05 = 575 \text{ грн}$$

Підрахуємо ефективний годинний фонд часу ПК за рік по формулі:

$$\text{Т}_{\text{ЕФ}} = (365 - 142 - 16) * 9 * 0,8 = 1490,4 \text{ год}$$

Розрахуємо витрати на електроенергію

$$\text{С}_{\text{ЕЛ}} = \text{ТЕФ} * Nc * \text{КЗ} * \text{ЦЕН} = 1490,4 * 0,26 * 1,75 * 0,954 = 646,94 \text{ грн}$$

Накладні витрати рівні:

$$\text{С}_{\text{Н}} = 10000 * 0,67 = 6700 \text{ грн.}$$

Отже експлуатаційні витрати(грн):

$$С_{ЕКС} = 34\ 481 + 7585{,}97 + 2875 + 575 + 646{,}94 + \ 6700 = 53\ 918{,}526 \text{ грн}$$

Тоді собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М-Г} = \frac{53\ 918{,}526}{1490{,}4} = 36{,}08 \text{ грн/год}$$

Витрати на машину для варіантів:

1) $С_м = 36{,}08 \ * 1174{,}432 = 43301{,}3078$
2) $С_м = 36{,}08 * \ 1564{,}344 = 56677{,}363$

Накладні витрати для варіантів:

1) $С_Н = 102\ 440 \ * 0{,}67 = \ 68634{,}8$
2) $С_Н = 136\ 567 \ * 0{,}67 = 91499{,}89$

Порахуємо повну вартість розробки для наших варіантів:

1) $С_{ПП} = 102\ 440 \ + 22536 + 43301{,}3078 + 68634{,}8 = 236912{,}108$
2) $С_{ПП} = 136\ 567 + 30044 + 56677{,}363 + 91499{,}89 = 314788{,}253$

Знайдемо коефіцієнт техніко економічного рівня використавши формулу:

$К_{ТЕРj} = К_{Кj} / С_{Фj}$

$К_{ТЕР1}$ = 6.9816/ 236912,108 = 0,00002947,

$К_{ТЕР2}$ = 6.27/ 314788,253 = 0,00001992.

5.5 Висновки

Отже, можемо зробити висновок, що перший варіант виявився більш ефективним.

ВИСНОВКИ

В даній дипломній роботі було створено реалізацію детектора аномальної активності комп’ютерних мереж з ціллю захисту комп’ютерної мережі від можливих загроз.

При написанні дипломної роботи були проаналізовані можливі варіанти розв’язання проблеми безпеки комп’ютерних систем і обгрунтовано вибір нейронних мереж у якості алгоритму для детектора аномальної активності.

Результатом роботи створоеного програмного продукту є відповідь для кожного елементу з вхідних даних на питання: чи є цей сигнал загрозою для комп’ютерної мережі. Результат роботи даного програмного продукту є цілком компетентним і дозволяє виявити загрозу з високою ймовірністю. Був проведений аналіз можливих покращень створеної реалізації.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. J.Rubina Parveen — Neural networks in cyber security — 2017. [Електронний ресурс] — Режим доступу: http://www.irjcs.com/volumes/vol4/iss09/08.SISPCS10095.pdf

2. Мустафаєв А. - Нейросетевая система виявлення комп'ютерних атак на основі аналізу мережевого трафіку - 2016. [Електронний ресурс] - Режим доступу: http://e-notabene.ru/nb/article_18834.html

3. Haibo Zhang, Qing Huang, Fangwei Li, Jiang Zhu — A network security situation prediction model based on wavelet neural network with optimized parameters — 2016. [Електронний ресурс] — Режим доступу: https://www.sciencedirect.com/science/article/pii/S2352864816300281

4. Min-Joo Kang, Je-Won Kang — Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security — 2016. [Електронний ресурс] — Режим доступу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4896428/pdf/pone.0155781.pdf

5. Д.К. Левоневского, Р.Р. Фаткіева - Розробка системи виявлення аномалій мережевого трафіку - 2014. [Електронний ресурс] - Режим доступу: https://journals.nstu.ru/vestnik/download_article?id=3366

6. Жигулін П.Е., Подворчан Д.В. - Аналіз мережевого трафіку за допомогою нейронних мереж - 2013. [Електронний ресурс] - Режим доступу: https://storage.tusur.ru/files/425/КИБЭВС-1005_Жигулин_П.В__Подворчан_Д.Э.pdf

7. "Halenar Igor, Juhasova Bohuslava, Juhas Martin, Nesticky Martin — Application of Neural Networks in Computer Security — 2013. [Електронний

ресурс] — Режим доступу: https://www.sciencedirect.com/science/article/pii/S1877705814003579

8. Талалаев А.А., Тищенко І.П., Фраленко В.П., Ємельянова Ю.Г. - Нейросетевая технологія виявлення мережевих атак на інформаційні ресурси — 2011. [Електронний ресурс] — Режим доступу: http://psta.psiras.ru/read/psta2011_3_3-15.pdf

9. E.Kesavulu Reddy — Neural Networks for Intrusion Detection and Its Applications — 2013. [Електронний ресурс] — Режим доступу: https://pdfs.semanticscholar.org/94f8/e1914ca526f53e9932890a0356394f9806f8.pdf

10. Балахонцев А.Ю., Сидорик Д.В., Сідоревич А.Н., Якутович М.В. - Нейросетевая система для виявлення атак в локальних обчислювальних мережах — 2004. [Електронний ресурс] — Режим доступу: http://elib.bsu.by/bitstream/123456789/179040/1/119-122.pdf

ДОДАТОК А

Лістинг програми


```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "from abc import ABCMeta, abstractmethod\n",
    "from math import sqrt\n",
    "from mayavi import mlab\n",
    "import operator\n",
    "import imageio\n",
    "from collections import OrderedDict\n",
    "from scipy.spatial.distance import euclidean\n",
    "from sklearn import preprocessing\n",
    "import csv\n",
    "import numpy as np\n",
    "import networkx as nx\n",
    "import re\n",
    "import os\n",
    "import shutil\n",
    "import sys\n",
    "import glob\n",
    "from past.builtins import xrange\n",
```

```
"from future.utils import iteritems\n",
"import time\n",
"\n",
"def sh(s):\n",
"    sum = 0\n",
"    for i, c in enumerate(s):\n",
"        sum += i * ord(c)\n",
"    return sum\n",
"\n",
"def create_data_graph(dots):\n",
"   \"\"\"Create the graph and returns the networkx version of it 'G'.\"\"\"\n",
"\n",
"    count = 0\n",
"\n",
"    G = nx.Graph()\n",
"\n",
"    for i in dots:\n",
"        G.add_node(count, pos=(i))\n",
"        count += 1\n",
"\n",
"    return G\n",
"\n",
"def get_ra(ra=0, ra_step=0.3):\n",
"    while True:\n",
"        if ra >= 360:\n",
"            ra = 0\n",
"        else:\n",
```

```
"            ra += ra_step\n",
"        yield ra\n",
"\n",
"def shrink_to_3d(data):\n",
"    result = []\n",
"\n",
"    for i in data:\n",
"        depth = len(i)\n",
"        if depth <= 3:\n",
"            result.append(i)\n",
"        else:\n",
"            sm = np.sum([(n) * v for n, v in enumerate(i[2:])])\n",
"            if sm == 0:\n",
"                sm = 1\n",
"\n",
"            r = np.array([i[0], i[1], i[2]])\n",
"            r *= sm\n",
"            r /= np.sum(r)\n",
"\n",
"            result.append(r)\n",
"\n",
"    return preprocessing.normalize(result, axis=0, norm='max')\n",
"\n",
"def draw_dots3d(dots, edges, fignum, clear=True,\n",
"                title='',\n",
"                size=(1024, 768), graph_colormap='viridis',\n",
"                bgcolor=(1, 1, 1),\n",
```

```
            "                node_color=(0.3, 0.65, 0.3), node_size=0.01,\n",
            "                edge_color=(0.3, 0.3, 0.9), edge_size=0.003,\n",
            "                  text_size=0.14, text_color=(0, 0, 0), text_coords=[0.84, 0.75],
text={},\n",
            "                title_size=0.3,\n",
            "                angle=get_ra()):\n",
            "\n",
            "      #  https://stackoverflow.com/questions/17751552/drawing-multiplex-
graphs-with-networkx\n",
            "\n",
            "    # numpy array of x, y, z positions in sorted node order\n",
            "    xyz = shrink_to_3d(dots)\n",
            "\n",
            "    if fignum == 0:\n",
            "                    mlab.figure(fignum,   bgcolor=bgcolor,   fgcolor=text_color,
size=size)\n",
            "\n",
            "    # Mayavi is buggy, and following code causes sockets leak.\n",
            "    #if mlab.options.offscreen:\n",
            "          #          mlab.figure(fignum,   bgcolor=bgcolor,   fgcolor=text_color,
size=size)\n",
            "    #elif fignum == 0:\n",
            "          #          mlab.figure(fignum,   bgcolor=bgcolor,   fgcolor=text_color,
size=size)\n",
            "\n",
            "    if clear:\n",
            "        mlab.clf()\n",
```

```
"\n",
"    # the x,y, and z co-ordinates are here\n",
"    # manipulate them to obtain the desired projection perspective\n",
"\n",
"    pts = mlab.points3d(xyz[:, 0], xyz[:, 1], xyz[:, 2],\n",
"                        scale_factor=node_size,\n",
"                        scale_mode='none',\n",
"                        color=node_color,\n",
"                        #colormap=graph_colormap,\n",
"                        resolution=20,\n",
"                        transparent=False)\n",
"\n",
"     mlab.text(text_coords[0], text_coords[1], '\\n'.join(['{} = {}'.format(n,
v) for n, v in text.items()]), width=text_size)\n",
"\n",
"    if clear:\n",
"        mlab.title(title, height=0.95)\n",
"        mlab.roll(next(angle))\n",
"        mlab.orientation_axes(pts)\n",
"        mlab.outline(pts)\n",
"\n",
"    \"\"\"\n",
"    for i, (x, y, z) in enumerate(xyz):\n",
"        label = mlab.text(x, y, str(i), z=z,\n",
"                          width=text_size, name=str(i), color=text_color)\n",
"        label.property.shadow = True\n",
"    \"\"\"\n",
```

```
        "\n",
        "    pts.mlab_source.dataset.lines = edges\n",
        "    tube = mlab.pipeline.tube(pts, tube_radius=edge_size)\n",
        "    mlab.pipeline.surface(tube, color=edge_color)\n",
        "\n",
        "    #mlab.show() # interactive window\n",
        "\n",
        "def draw_graph3d(graph, fignum, *args, **kwargs):\n",
        "    graph_pos = nx.get_node_attributes(graph, 'pos')\n",
        "    edges = np.array([e for e in graph.edges()])\n",
        "            dots = np.array([graph_pos[v] for v in sorted(graph)],
dtype='float64')\n",
        "\n",
        "    draw_dots3d(dots, edges, fignum, *args, **kwargs)\n",
        "\n",
        "def generate_host_activity(is_normal):\n",
        "    # Host loads is changed only in 25% cases.\n",
        "    attack_percent = 25\n",
        "    up_level = (20, 30)\n",
        "\n",
        "    # CPU load in percent.\n",
        "    cpu_load = (10, 30)\n",
        "    # Disk IO per second.\n",
        "    iops = (10, 50)\n",
        "    # Memory consumption in percent.\n",
        "    mem_cons = (30, 60)\n",
        "    # Memory consumption in Mb/s.\n",
```

```
"    netw_act = (10, 50)\n",
"\n",
"    cur_up_level = 0\n",
"\n",
"    if not is_normal and np.random.randint(0, 100) < attack_percent:\n",
"        cur_up_level = np.random.randint(*up_level)\n",
"\n",
"            cpu_load = np.random.randint(cur_up_level + cpu_load[0], cur_up_level + cpu_load[1])\n",
"        iops = np.random.randint(cur_up_level + iops[0], cur_up_level + iops[1])\n",
"          mem_cons = np.random.randint(cur_up_level + mem_cons[0], cur_up_level + mem_cons[1])\n",
"            netw_act = np.random.randint(cur_up_level + netw_act[0], cur_up_level + netw_act[1])\n",
"\n",
"    return cpu_load, iops, mem_cons, netw_act\n",
"\n",
"def read_ids_data(data_file, activity_type='normal', labels_file='NSL_KDD/Field Names.csv', with_host=False):\n",
"        selected_parameters = ['duration', 'protocol_type', 'service', 'flag', 'src_bytes', 'dst_bytes', 'land', 'wrong_fragment', 'urgent', 'serror_rate',\n",
"                               'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_srv_count', 'count']\n",
"\n",
"    # \"Label\" - \"converter function\" dictionary.\n",
"    label_dict = OrderedDict()\n",
```

```
"    result = []\n",
"\n",
"    with open(labels_file) as lf:\n",
"        labels = csv.reader(lf)\n",
"        for label in labels:\n",
"            if len(label) == 1 or label[1] == 'continuous':\n",
"                label_dict[label[0]] = lambda l: np.float64(l)\n",
"            elif label[1] == 'symbolic':\n",
"                label_dict[label[0]] = lambda l: sh(l)\n",
"\n",
"    f_list = [i for i in label_dict.values()]\n",
"    n_list = [i for i in label_dict.keys()]\n",
"\n",
"    if activity_type == 'normal':\n",
"        data_type = lambda t: t == 'normal'\n",
"    elif activity_type == 'abnormal':\n",
"        data_type = lambda t: t != 'normal'\n",
"    elif activity_type == 'full':\n",
"        data_type = lambda t: True\n",
"    else:\n",
"        raise ValueError('`activity_type` must be \"normal\", \"abnormal\" or \"full\"')\n",
"\n",
"     print('Reading {} activity from the file \"{}\" [generated host data {} included]...'.\n",
"         format(activity_type, data_file, 'was' if with_host else 'was not'))\n",
"                    anomaly_level = 0\n",
```

```
"            else:\n",
"                normal_records_counter += 1\n",
"\n",
"            if i % save_step == 0:\n",
"                tm = time.time() - start_time\n",
"                 print('Abnormal records = {}, Normal records = {}, Detection time = {} s, Time per record = {} s'.\n",
"                       format(anomaly_records_counter, normal_records_counter, round(tm, 2), tm / i if i else 0))\n",
"\n",
"        tm = time.time() - start_time\n",
"\n",
"        print('{} [abnormal records = {}, normal records = {}, detection time = {} s, time per record = {} s]'.\n",
"                                    format('Anomalies were detected (count = {})'.format(anomalies_counter) if anomalies_counter else 'Anomalies weren\\'t detected',\n",
"                anomaly_records_counter, normal_records_counter, round(tm, 2), tm / len(data)))\n",
"\n",
"        return anomalies_counter > 0\n",
"\n",
"    def test_node(self, node, train=False):\n",
"        n, dist = self._determine_closest_vertice(node)\n",
"        dev = self._calculate_deviation_params()\n",
"        dev = dev.get(frozenset(nx.node_connected_component(self._graph, n)), dist + 1)\n",
```

```
"        dist_sub_dev = dist - dev\n",
"        if dist_sub_dev > 0:\n",
"            return dist_sub_dev\n",
"\n",
"        if train:\n",
"            self._dev_params = None\n",
"            self._train_on_data_item(node)\n",
"\n",
"        return 0\n",
"\n",
"    @abstractmethod\n",
"    def _train_on_data_item(self, data_item):\n",
"        raise NotImplementedError()\n",
"\n",
"    @abstractmethod\n",
"    def _save_img(self, fignum, training_step):\n",
"        \"\"\".\"\"\"\n",
"        raise NotImplementedError()\n",
"\n",
"                                    def          _calculate_deviation_params(self,
distance_function_params={}):\n",
"        if self._dev_params is not None:\n",
"            return self._dev_params\n",
"\n",
"        clusters = {}\n",
"        dcvd = self._determine_closest_vertice\n",
"        dlen = len(self._data)\n",
```

```
"        #dmean = np.mean(self._data, axis=1)\n",
"        #deviation = 0\n",
"\n",
"        for node in self._data:\n",
"            n = dcvd(node, **distance_function_params)\n",
"                                                    cluster =
clusters.setdefault(frozenset(nx.node_connected_component(self._graph, n[0])), [0,
0])\n",
"            cluster[0] += n[1]\n",
"            cluster[1] += 1\n",
"\n",
"        clusters = {k: sqrt(v[0]/v[1]) for k, v in clusters.items()}\n",
"\n",
"        self._dev_params = clusters\n",
"\n",
"        return clusters\n",
"\n",
"    def _determine_closest_vertice(self, curnode):\n",
"        \"\"\".\"\"\"\n",
"\n",
"        pos = nx.get_node_attributes(self._graph, 'pos')\n",
"        kv =list(zip(*pos.items()))\n",
"        distances = np.linalg.norm(kv[1] - curnode, ord=2, axis=1)\n",
"\n",
"        i0 = np.argsort(distances)[0]\n",
"\n",
"        return kv[0][i0], distances[i0]\n",
```

```
        "\n",
        "    def _determine_2closest_vertices(self, curnode):\n",
        "        \"\"\"Where this curnode is actually the x,y index of the data we want to analyze.\"\"\"\n",
        "\n",
        "        pos = nx.get_node_attributes(self._graph, 'pos')\n",
        "        l_pos = len(pos)\n",
        "        if l_pos == 0:\n",
        "            return None, None\n",
        "        elif l_pos == 1:\n",
        "            return pos[0], None\n",
        "\n",
        "        kv = list(zip(*pos.items()))\n",
        "        # Calculate Euclidean distance (2-norm of difference vectors) and get first two indexes of the sorted array.\n",
        "        # Or a Euclidean-closest nodes index.\n",
        "        distances = np.linalg.norm(kv[1] - curnode, ord=2, axis=1)\n",
        "        i0, i1 = np.argsort(distances)[0:2]\n",
        "\n",
        "        winner1 = tuple((kv[0][i0], distances[i0]))\n",
        "        winner2 = tuple((kv[0][i1], distances[i1]))\n",
        "\n",
        "        return winner1, winner2\n",
        "\n",
        "class IGNG(NeuralGas):\n",
        "            \"\"\"Incremental Growing Neural Gas multidimensional implementation\"\"\"\n",
```

```
        "\n",
        "            def __init__(self, data, surface_graph=None, eps_b=0.05,
eps_n=0.0005, max_age=10,\n",
        "              a_mature=1, output_images_dir='images'):\n",
        "       \"\"\".\"\"\"\n",
        "\n",
        "       NeuralGas.__init__(self, data, surface_graph, output_images_dir)\n",
        "\n",
        "       self._eps_b = eps_b\n",
        "       self._eps_n = eps_n\n",
        "       self._max_age = max_age\n",
        "       self._a_mature = a_mature\n",
        "       self._num_of_input_signals = 0\n",
        "       self._fignum = 0\n",
        "       self._max_train_iters = 0\n",
        "\n",
        "       # Initial value is a standard deviation of the data.\n",
        "       self._d = np.std(data)\n",
        "\n",
        "    def train(self, max_iterations=100, save_step=0):\n",
        "       \"\"\"IGNG training method\"\"\"\n",
        "\n",
        "       self._dev_params = None\n",
        "       self._max_train_iters = max_iterations\n",
        "\n",
        "       fignum = self._fignum\n",
        "       self._save_img(fignum, 0)\n",
```

```
        "       CHS = self.__calinski_harabaz_score\n",
        "       igng = self.__igng\n",
        "       data = self._data\n",
        "\n",
        "       if save_step < 1:\n",
        "           save_step = max_iterations\n",
        "\n",
        "       old = 0\n",
        "       calin = CHS()\n",
        "       i_count = 0\n",
        "       start_time = self._start_time = time.time()\n",
        "\n",
        "       while old - calin <= 0:\n",
        "           print('Iteration {0:d}...'.format(i_count))\n",
        "           i_count += 1\n",
        "           steps = 1\n",
        "           while steps <= max_iterations:\n",
        "               for i, x in enumerate(data):\n",
        "                   igng(x)\n",
        "                   if i % save_step == 0:\n",
        "                       tm = time.time() - start_time\n",
        "                     print('Training time = {} s, Time per record = {} s, Training
step = {}, Clusters count = {}, Neurons = {}, CHI = {}'.\n",
        "                           format(round(tm, 2),\n",
        "                                  tm / (i if i and i_count == 0 else len(data)),\n",
        "                                  i_count,\n",
        "                                  self.number_of_clusters(),\n",
```

```
"                               len(self._graph),\n",
"                               old - calin)\n",
"                           )\n",
"                       self._save_img(fignum, i_count)\n",
"                       fignum += 1\n",
"                steps += 1\n",
"\n",
"            self._d -= 0.1 * self._d\n",
"            old = calin\n",
"            calin = CHS()\n",
"            print('Training complete, clusters count = {}, training time = {}
s'.format(self.number_of_clusters(), round(time.time() - start_time, 2)))\n",
"        self._fignum = fignum\n",
"\n",
"    def _train_on_data_item(self, data_item):\n",
"        steps = 0\n",
"        igng = self.__igng\n",
"\n",
"        # while steps < self._max_train_iters:\n",
"        while steps < 5:\n",
"            igng(data_item)\n",
"            steps += 1\n",
"\n",
"    def __long_train_on_data_item(self, data_item):\n",
"        \"\"\".\"\"\"\n",
"\n",
"        np.append(self._data, data_item)\n",
```

```
        "\n",
        "        self._dev_params = None\n",
        "        CHS = self.__calinski_harabaz_score\n",
        "        igng = self.__igng\n",
        "        data = self._data\n",
        "\n",
        "        max_iterations = self._max_train_iters\n",
        "\n",
        "        old = 0\n",
        "        calin = CHS()\n",
        "        i_count = 0\n",
        "\n",
        "        # Strictly less.\n",
        "        while old - calin < 0:\n",
        "                                print('Training with new normal node, step
{0:d}...'.format(i_count))\n",
        "            i_count += 1\n",
        "            steps = 0\n",
        "\n",
        "            if i_count > 100:\n",
        "                print('BUG', old, calin)\n",
        "                break\n",
        "\n",
        "            while steps < max_iterations:\n",
        "                igng(data_item)\n",
        "                steps += 1\n",
        "            self._d -= 0.1 * self._d\n",
```

```
"            old = calin\n",
"            calin = CHS()\n",
"\n",
"    def _calculate_deviation_params(self, skip_embryo=True):\n",
"                                                    return super(IGNG,
self)._calculate_deviation_params(distance_function_params={'skip_embryo':
skip_embryo})\n",
"\n",
"    def __calinski_harabaz_score(self, skip_embryo=True):\n",
"        graph = self._graph\n",
"        nodes = graph.nodes\n",
"        extra_disp, intra_disp = 0., 0.\n",
"\n",
"        # CHI = [B / (c - 1)]/[W / (n - c)]\n",
"        # Total numb er of neurons.\n",
"        #ns = nx.get_node_attributes(self._graph, 'n_type')\n",
"        c = len([v for v in nodes.values() if v['n_type'] == 1]) if skip_embryo
else len(nodes)\n",
"        # Total number of data.\n",
"        n = len(self._data)\n",
"\n",
"        # Mean of the all data.\n",
"        mean = np.mean(self._data, axis=1)\n",
"\n",
"        pos = nx.get_node_attributes(self._graph, 'pos')\n",
"\n",
"        for node, k in pos.items():\n",
```

```
"            if skip_embryo and nodes[node]['n_type'] == 0:\n",
"                # Skip embryo neurons.\n",
"                continue\n",
"\n",
"            mean_k = np.mean(k)\n",
"            extra_disp += len(k) * np.sum((mean_k - mean) ** 2)\n",
"            intra_disp += np.sum((k - mean_k) ** 2)\n",
"\n",
"        return (1. if intra_disp == 0. else\n",
"                extra_disp * (n - c) /\n",
"                (intra_disp * (c - 1.)))\n",
"\n",
"    def _determine_closest_vertice(self, curnode, skip_embryo=True):\n",
"        \"\"\"Where this curnode is actually the x,y index of the data we want
to analyze.\"\"\"\n",
"\n",
"        pos = nx.get_node_attributes(self._graph, 'pos')\n",
"        nodes = self._graph.nodes\n",
"\n",
"        distance = sys.maxint\n",
"        for node, position in pos.items():\n",
"            if skip_embryo and nodes[node]['n_type'] == 0:\n",
"                # Skip embryo neurons.\n",
"                continue\n",
"            dist = euclidean(curnode, position)\n",
"            if dist < distance:\n",
"                distance = dist\n",
```

```
        "\n",
        "        return node, distance\n",
        "\n",
        "    def __get_specific_nodes(self, n_type):\n",
        "                return [n for n, p in nx.get_node_attributes(self._graph,
'n_type').items() if p == n_type]\n",
        "\n",
        "    def __igng(self, cur_node):\n",
        "        \"\"\"Main IGNG training subroutine\"\"\"\n",
        "\n",
        "        # find nearest unit and second nearest unit\n",
        "        winner1, winner2 = self._determine_2closest_vertices(cur_node)\n",
        "        graph = self._graph\n",
        "        nodes = graph.nodes\n",
        "        d = self._d\n",
        "\n",
        "        # Second list element is a distance.\n",
        "        if winner1 is None or winner1[1] >= d:\n",
        "            # 0 - is an embryo type.\n",
        "            graph.add_node(self._count, pos=cur_node, n_type=0, age=0)\n",
        "            winner_node1 = self._count\n",
        "            self._count += 1\n",
        "            return\n",
        "        else:\n",
        "            winner_node1 = winner1[0]\n",
        "\n",
        "        # Second list element is a distance.\n",
```

```
"        if winner2 is None or winner2[1] >= d:\n",
"            # 0 - is an embryo type.\n",
"            graph.add_node(self._count, pos=cur_node, n_type=0, age=0)\n",
"            winner_node2 = self._count\n",
"            self._count += 1\n",
"            graph.add_edge(winner_node1, winner_node2, age=0)\n",
"            return\n",
"        else:\n",
"            winner_node2 = winner2[0]\n",
"\n",
"        # Increment the age of all edges, emanating from the winner.\n",
"        for e in graph.edges(winner_node1, data=True):\n",
"            e[2]['age'] += 1\n",
"\n",
"        w_node = nodes[winner_node1]\n",
"        # Move the winner node towards current node.\n",
"        w_node['pos'] += self._eps_b * (cur_node - w_node['pos'])\n",
"\n",
"        neighbors = nx.all_neighbors(graph, winner_node1)\n",
"        a_mature = self._a_mature\n",
"\n",
"        for n in neighbors:\n",
"            c_node = nodes[n]\n",
"            # Move all direct neighbors of the winner.\n",
"            c_node['pos'] += self._eps_n * (cur_node - c_node['pos'])\n",
"            # Increment the age of all direct neighbors of the winner.\n",
"            c_node['age'] += 1\n",
```

```
"            if c_node['n_type'] == 0 and c_node['age'] >= a_mature:\n",
"                # Now, it's a mature neuron.\n",
"                c_node['n_type'] = 1\n",
"\n",
"        # Create connection with age == 0 between two winners.\n",
"        graph.add_edge(winner_node1, winner_node2, age=0)\n",
"\n",
"        max_age = self._max_age\n",
"\n",
"       # If there are ages more than maximum allowed age, remove them.\n",
"        age_of_edges = nx.get_edge_attributes(graph, 'age')\n",
"        for edge, age in iteritems(age_of_edges):\n",
"            if age >= max_age:\n",
"                graph.remove_edge(edge[0], edge[1])\n",
"\n",
"        # If it causes isolated vertix, remove that vertex as well.\n",
"        #graph.remove_nodes_from(nx.isolates(graph))\n",
"        for node, v in nodes.items():\n",
"            if v['n_type'] == 0:\n",
"                # Skip embryo neurons.\n",
"                continue\n",
"            if not graph.neighbors(node):\n",
"                graph.remove_node(node)\n",
"\n",
"    def _save_img(self, fignum, training_step):\n",
"        \"\"\".\"\"\"\n",
"\n",
```

```
"          title='Incremental Growing Neural Gas for the network anomalies detection'\n",
"\n",
"        if self._surface_graph is not None:\n",
"            text = OrderedDict([\n",
"                ('Image', fignum),\n",
"                ('Training step', training_step),\n",
"             ('Time', '{} s'.format(round(time.time() - self._start_time, 2))),\n",
"                ('Clusters count', self.number_of_clusters()),\n",
"                ('Neurons', len(self._graph)),\n",
"                ('    Mature', len(self.__get_specific_nodes(1))),\n",
"                ('    Embryo', len(self.__get_specific_nodes(0))),\n",
"                ('Connections', len(self._graph.edges)),\n",
"                ('Data records', len(self._data))\n",
"            ])\n",
"\n",
"            draw_graph3d(self._surface_graph, fignum, title=title)\n",
"\n",
"        graph = self._graph\n",
"\n",
"        if len(graph) > 0:\n",
"            #graph_pos = nx.get_node_attributes(graph, 'pos')\n",
"            #nodes = sorted(self.get_specific_nodes(1))\n",
"           #dots = np.array([graph_pos[v] for v in nodes], dtype='float64')\n",
"             #edges = np.array([e for e in graph.edges(nodes) if e[0] in nodes and e[1] in nodes])\n",
```

```
"            #draw_dots3d(dots, edges, fignum, clear=False, node_color=(1, 0,
0))\n",
"\n",
"             draw_graph3d(graph, fignum, clear=False, node_color=(1, 0, 0),
title=title,\n",
"                          text=text)\n",
"\n",
"                 mlab.savefig(\"{0}/{1}.png\".format(self._output_images_dir,
str(fignum)))\n",
"        #mlab.close(fignum)\n",
"\n",
"class GNG(NeuralGas):\n",
"    \"\"\"Growing Neural Gas multidimensional implementation\"\"\"\n",
"\n",
"              def __init__(self, data, surface_graph=None, eps_b=0.05,
eps_n=0.0006, max_age=15,\n",
"                lambda_=20, alpha=0.5, d=0.005, max_nodes=1000,\n",
"                output_images_dir='images'):\n",
"        \"\"\".\"\"\"\n",
"       NeuralGas.__init__(self, data, surface_graph, output_images_dir)\n",
"\n",
"        self._eps_b = eps_b\n",
"        self._eps_n = eps_n\n",
"        self._max_age = max_age\n",
"        self._lambda = lambda_\n",
"        self._alpha = alpha\n",
"        self._d = d\n",
```

```
        "       self._max_nodes = max_nodes\n",
        "       self._fignum = 0\n",
        "\n",
        "       self.__add_initial_nodes()\n",
        "\n",
        "                   def train(self, max_iterations=10000, save_step=50,
stop_on_chi=False):\n",
        "       \"\"\".\"\"\"\n",
        "\n",
        "       self._dev_params = None\n",
        "       self._save_img(self._fignum, 0)\n",
        "       graph = self._graph\n",
        "       max_nodes = self._max_nodes\n",
        "       d = self._d\n",
        "       ld = self._lambda\n",
        "       alpha = self._alpha\n",
        "       update_winner = self.__update_winner\n",
        "       data = self._data\n",
        "       CHS = self.__calinski_harabaz_score\n",
        "       old = 0\n",
        "       calin = CHS()\n",
        "       start_time = self._start_time = time.time()\n",
        "       train_step = self.__train_step\n",
        "\n",
        "       for i in xrange(1, max_iterations):\n",
        "           tm = time.time() - start_time\n",
```

```
"           print('Training time = {} s, Time per record = {} s, Training step = {}/{}, Clusters count = {}, Neurons = {}'.\n",
"                 format(round(tm, 2),\n",
"                     tm / len(data),\n",
"                     i, max_iterations,\n",
"                     self.number_of_clusters(),\n",
"                     len(self._graph))\n",
"                 )\n",
"\n",
"           for x in data:\n",
"               update_winner(x)\n",
"\n",
"               train_step(i, alpha, ld, d, max_nodes, True, save_step, graph, update_winner)\n",
"\n",
"           old = calin\n",
"           calin = CHS()\n",
"\n",
"           # Stop on the enough clusterization quality.\n",
"           if stop_on_chi and old - calin > 0:\n",
"               break\n",
"           print('Training complete, clusters count = {}, training time = {} s'.format(self.number_of_clusters(), round(time.time() - start_time, 2)))\n",
"\n",
"    def __train_step(self, i, alpha, ld, d, max_nodes, save_img, save_step, graph, update_winner):\n",
"        g_nodes = graph.nodes\n",
```

```
        "\n",
        "       # Step 8: if number of input signals generated so far\n",
        "       if i % ld == 0 and len(graph) < max_nodes:\n",
        "          # Find a node with the largest error.\n",
        "          errorvectors = nx.get_node_attributes(graph, 'error')\n",
        "                                  node_largest_error   =   max(errorvectors.items(),
key=operator.itemgetter(1))[0]\n",
        "\n",
        "          # Find a node from neighbor of the node just found, with a largest
error.\n",
        "          neighbors = graph.neighbors(node_largest_error)\n",
        "          max_error_neighbor = None\n",
        "          max_error = -1\n",
        "\n",
        "          for n in neighbors:\n",
        "             ce = g_nodes[n]['error']\n",
        "             if ce > max_error:\n",
        "                max_error = ce\n",
        "                max_error_neighbor = n\n",
        "\n",
        "          # Decrease error variable of other two nodes by multiplying with
alpha.\n",
        "          new_max_error = alpha * errorvectors[node_largest_error]\n",
        "          graph.nodes[node_largest_error]['error'] = new_max_error\n",
        "          graph.nodes[max_error_neighbor]['error'] = alpha * max_error\n",
        "\n",
        "          # Insert a new unit half way between these two.\n",
```

```
            "            self._count += 1\n",
            "            new_node = self._count\n",
            "            graph.add_node(new_node,\n",
            "
pos=self.__get_average_dist(g_nodes[node_largest_error]['pos'],
g_nodes[max_error_neighbor]['pos']),\n",
            "                        error=new_max_error)\n",
            "\n",
            "            # Insert edges between new node and other two nodes.\n",
            "            graph.add_edge(new_node, max_error_neighbor, age=0)\n",
            "            graph.add_edge(new_node, node_largest_error, age=0)\n",
            "\n",
            "            # Remove edge between old nodes.\n",
            "            graph.remove_edge(max_error_neighbor, node_largest_error)\n",
            "\n",
            "            if True and i % save_step == 0:\n",
            "                self._fignum += 1\n",
            "                self._save_img(self._fignum, i)\n",
            "\n",
            "            # step 9: Decrease all error variables.\n",
            "            for n in graph.nodes():\n",
            "                oe = g_nodes[n]['error']\n",
            "                g_nodes[n]['error'] -= d * oe\n",
            "\n",
            "    def _train_on_data_item(self, data_item):\n",
            "        \"\"\"IGNG training method\"\"\"\n",
            "\n",
```

```
"       np.append(self._data, data_item)\n",
"\n",
"       graph = self._graph\n",
"       max_nodes = self._max_nodes\n",
"       d = self._d\n",
"       ld = self._lambda\n",
"       alpha = self._alpha\n",
"       update_winner = self.__update_winner\n",
"       data = self._data\n",
"       train_step = self.__train_step\n",
"\n",
"       #for i in xrange(1, 5):\n",
"       update_winner(data_item)\n",
"               train_step(0, alpha, ld, d, max_nodes, False, -1, graph,
update_winner)\n",
"\n",
"   def _calculate_deviation_params(self):\n",
"       return super(GNG, self)._calculate_deviation_params()\n",
"\n",
"   def __add_initial_nodes(self):\n",
"       \"\"\"Initialize here\"\"\"\n",
"\n",
"       node1 = self._data[np.random.randint(0, len(self._data))]\n",
"       node2 = self._data[np.random.randint(0, len(self._data))]\n",
"\n",
"       # make sure you dont select same positions\n",
"       if self.__is_nodes_equal(node1, node2):\n",
```

```
"         raise ValueError(\"Rerun ----------> similar nodes selected\")\n",
"\n",
"        self._count = 0\n",
"        self._graph.add_node(self._count, pos=node1, error=0)\n",
"        self._count += 1\n",
"        self._graph.add_node(self._count, pos=node2, error=0)\n",
"        self._graph.add_edge(self._count - 1, self._count, age=0)\n",
"\n",
"    def __is_nodes_equal(self, n1, n2):\n",
"        return len(set(n1) & set(n2)) == len(n1)\n",
"\n",
"    def __update_winner(self, curnode):\n",
"        \"\"\".\"\"\"\n",
"\n",
"        # find nearest unit and second nearest unit\n",
"        winner1, winner2 = self._determine_2closest_vertices(curnode)\n",
"        winner_node1 = winner1[0]\n",
"        winner_node2 = winner2[0]\n",
"        win_dist_from_node = winner1[1]\n",
"        graph = self._graph\n",
"        g_nodes = graph.nodes\n",
"\n",
"        # Update the winner error.\n",
"        g_nodes[winner_node1]['error'] +=  + win_dist_from_node**2\n",
"\n",
"        # Move the winner node towards current node.\n",
```

```
"                g_nodes[winner_node1]['pos'] += self._eps_b * (curnode - g_nodes[winner_node1]['pos'])\n",
"\n",
"        eps_n = self._eps_n\n",
"\n",
"        # Now update all the neighbors distances.\n",
"        for n in nx.all_neighbors(graph, winner_node1):\n",
"            g_nodes[n]['pos'] += eps_n * (curnode - g_nodes[n]['pos'])\n",
"\n",
"        # Update age of the edges, emanating from the winner.\n",
"        for e in graph.edges(winner_node1, data=True):\n",
"            e[2]['age'] += 1\n",
"\n",
"        # Create or zeroe edge between two winner nodes.\n",
"        graph.add_edge(winner_node1, winner_node2, age=0)\n",
"\n",
"       # if there are ages more than maximum allowed age, remove them\n",
"        age_of_edges = nx.get_edge_attributes(graph, 'age')\n",
"        max_age = self._max_age\n",
"        for edge, age in age_of_edges.items():\n",
"            if age >= max_age:\n",
"                graph.remove_edge(edge[0], edge[1])\n",
"\n",
"        # If it causes isolated vertix, remove that vertex as well.\n",
"        for node in g_nodes:\n",
"            if not graph.neighbors(node):\n",
"                graph.remove_node(node)\n",
```

```
"\n",
"    def __get_average_dist(self, a, b):\n",
"        \"\"\".\"\"\"\n",
"\n",
"        return (a + b) / 2\n",
"\n",
"    def __calinski_harabaz_score(self):\n",
"        graph = self._graph\n",
"        nodes = graph.nodes\n",
"        extra_disp, intra_disp = 0., 0.\n",
"\n",
"        # CHI = [B / (c - 1)]/[W / (n - c)]\n",
"        # Total numb er of neurons.\n",
"        #ns = nx.get_node_attributes(self._graph, 'n_type')\n",
"        c = len(nodes)\n",
"        # Total number of data.\n",
"        n = len(self._data)\n",
"\n",
"        # Mean of the all data.\n",
"        mean = np.mean(self._data, axis=1)\n",
"\n",
"        pos = nx.get_node_attributes(self._graph, 'pos')\n",
"\n",
"        for node, k in pos.items():\n",
"            mean_k = np.mean(k)\n",
"            extra_disp += len(k) * np.sum((mean_k - mean) ** 2)\n",
"            intra_disp += np.sum((k - mean_k) ** 2)\n",
```

```
"\n",
"    def _save_img(self, fignum, training_step):\n",
"        \"\"\".\"\"\"\n",
"\n",
"        title = 'Growing Neural Gas for the network anomalies detection'\n",
"\n",
"        if self._surface_graph is not None:\n",
"            text = OrderedDict([\n",
"                ('Image', fignum),\n",
"                ('Training step', training_step),\n",
"             ('Time', '{} s'.format(round(time.time() - self._start_time, 2))),\n",
"                ('Clusters count', self.number_of_clusters()),\n",
"                ('Neurons', len(self._graph)),\n",
"                ('Connections', len(self._graph.edges)),\n",
"                ('Data records', len(self._data))\n",
"            ])\n",
"\n",
"            draw_graph3d(self._surface_graph, fignum, title=title)\n",
"\n",
"        graph = self._graph\n",
"\n",
"        if len(graph) > 0:\n",
"         draw_graph3d(graph, fignum, clear=False, node_color=(1, 0, 0),\n",
"                      title=title,\n",
"                      text=text)\n",
"\n",
"\n",
```

```
"    mlab.options.offscreen = True\n",
"        test_detector(use_hosts_data=False, max_iters=7000, alg=GNG, output_gif='gng_wohosts.gif')\n",
"    print('Working time = {}'.format(round(time.time() - start_time, 2)))\n",
"        test_detector(use_hosts_data=True, max_iters=7000, alg=GNG, output_gif='gng_whosts.gif')\n",
"    print('Working time = {}'.format(round(time.time() - start_time, 2)))\n",
"        test_detector(use_hosts_data=False, max_iters=100, alg=IGNG, output_gif='igng_wohosts.gif')\n",
"    print('Working time = {}'.format(round(time.time() - start_time, 2)))\n",
"        test_detector(use_hosts_data=True, max_iters=100, alg=IGNG, output_gif='igng_whosts.gif')\n",
"     print('Full working time = {}'.format(round(time.time() - start_time, 2)))\n",
"\n",
"    return 0\n",
"\n",
"if __name__ == \"__main__\":\n",
"    exit(main())"
]
},
{
"cell_type": "code",
"execution_count": null,
"metadata": {},
"outputs": [],
"source": []
```

```
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": []
 }
],
"metadata": {
 "kernelspec": {
  "display_name": "Python (pyconda)",
  "language": "python",
  "name": "pyconda"
 },
 "language_info": {
  "codemirror_mode": {
   "name": "ipython",
   "version": 3
  },
  "file_extension": ".py",
  "mimetype": "text/x-python",
  "name": "python",
  "nbconvert_exporter": "python",
  "pygments_lexer": "ipython3",
  "version": "3.7.3"
 }
```

},
"nbformat": 4,
"nbformat_minor": 2
}

ДОДАТОК Б

Ілюстративний матеріал

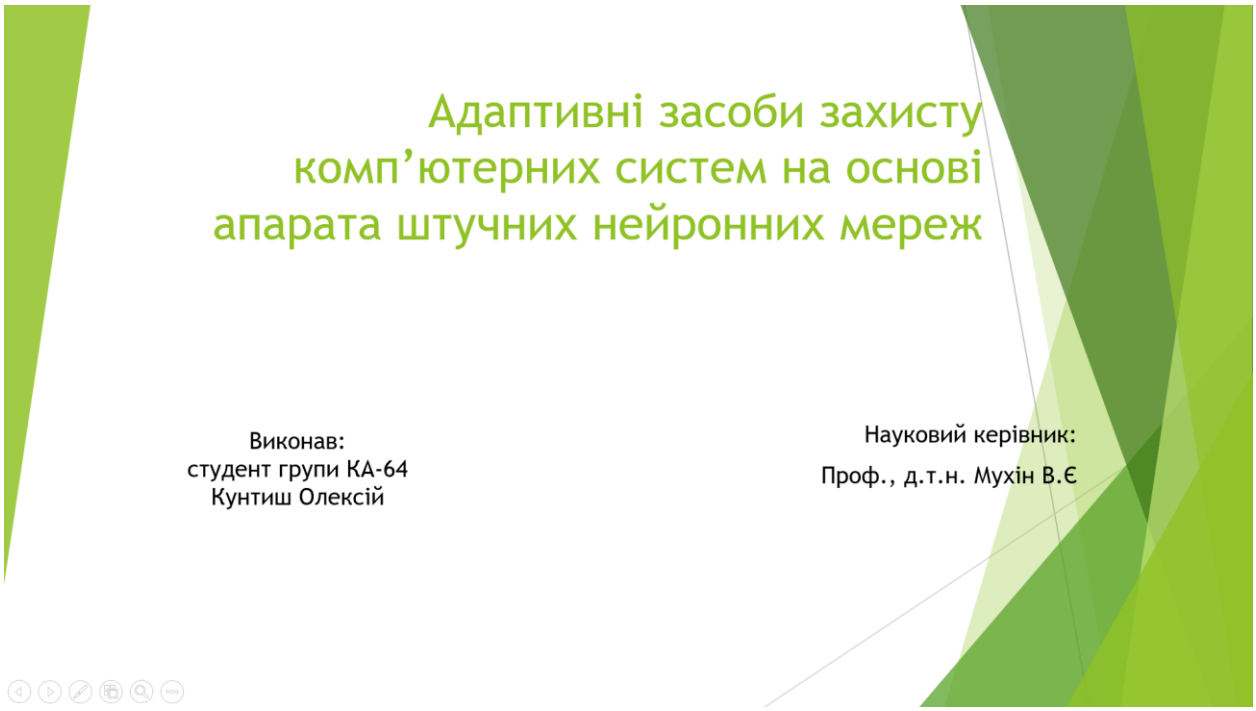

Рисунок 1 - Вступний слайд

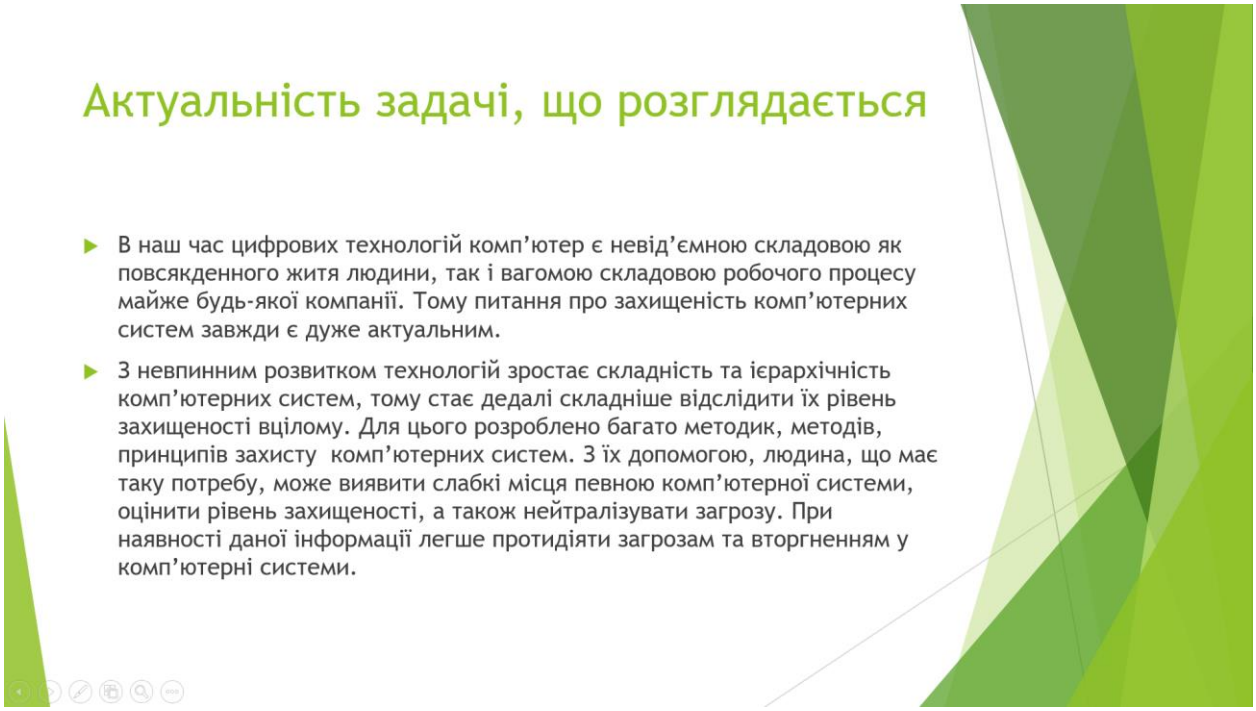

Рисунок 2 - Актуальність задачі

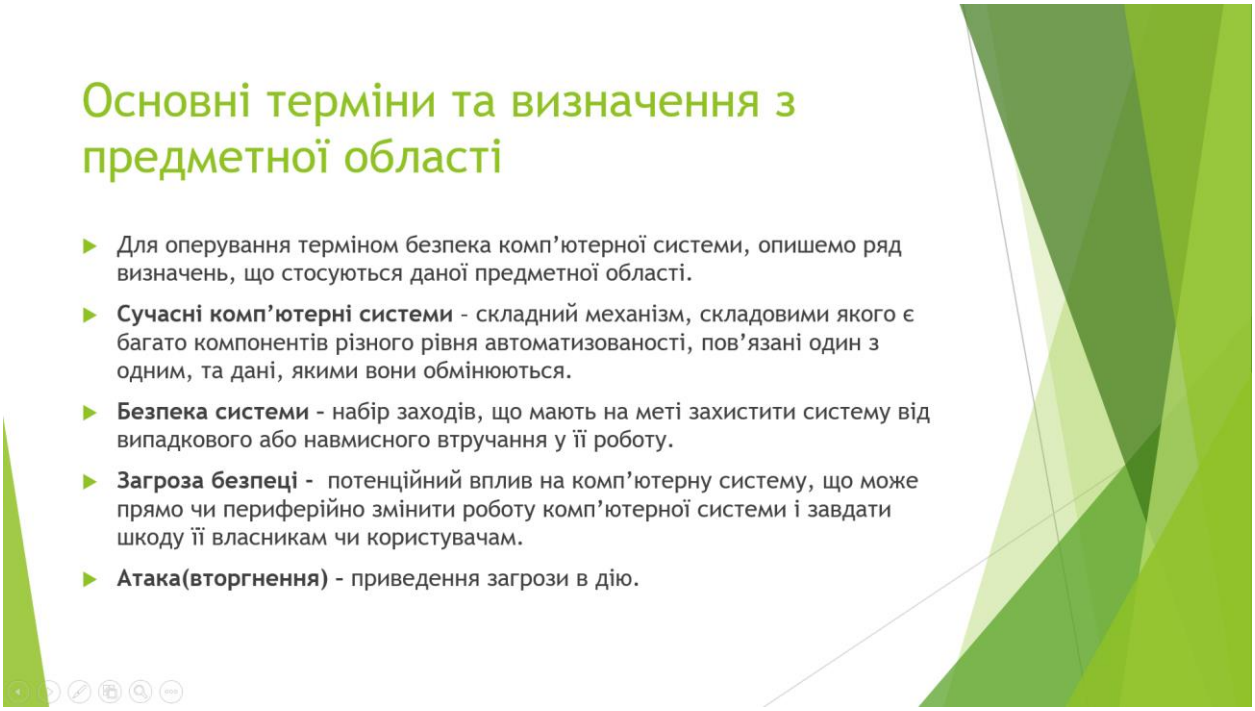

Рисунок 3 - Основні терміни та визначення

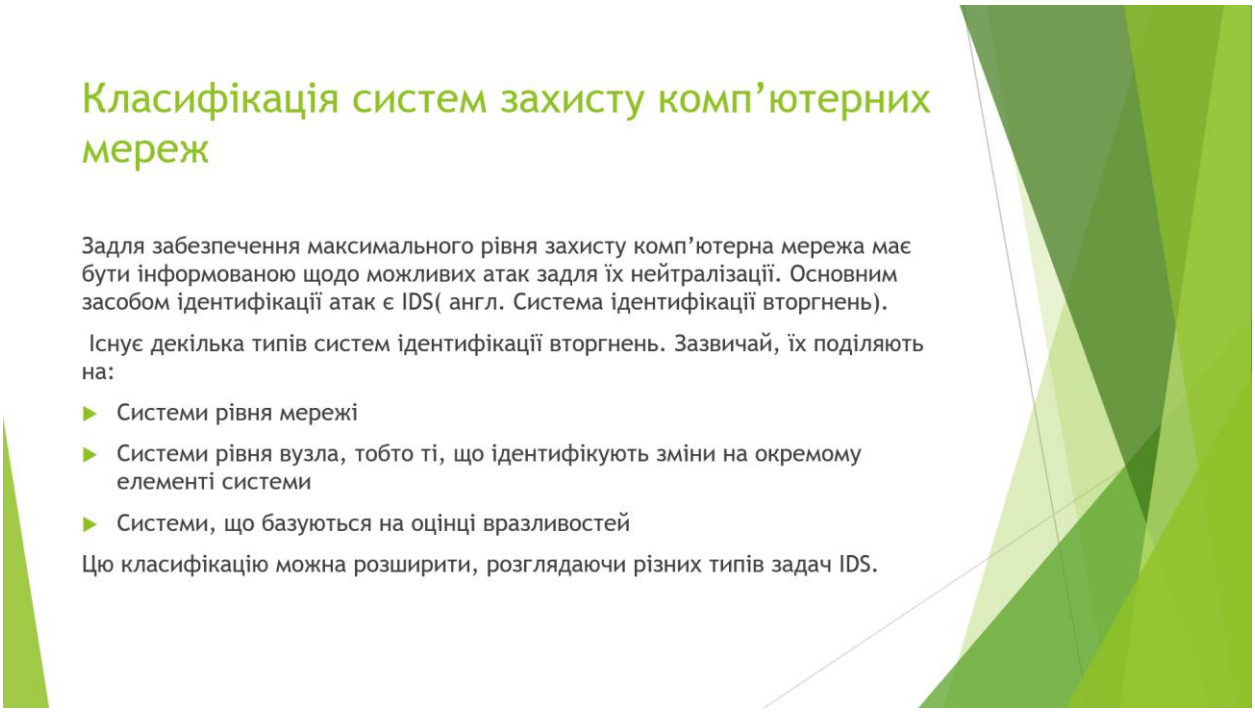

Рисунок 4 - Класифікація систем захисту комп’ютерних мереж

Рисунок 5 - Схема роботи звичайної IDS

Рисунок 6 - Порівняльний аналіз існуючих рішень

Доцільність застосування НМ для виявлення вторгнень

- Аналіз теоретичних робіт вказує на те, що в загальному випадку для КС принципова доцільність застосування нейромережевих методів розпізнавання визначається:
- 1. Відсутністю або низькою ефективністю явних алгоритмічних методів розпізнавання.
- 2. Можливістю навчання НМ.
- 3. Можливістю забезпечення технічних аспектів використання НМ.

Рисунок 7 - Доцільність застосувння НМ

Алгоритм зростаючого нейронного газу

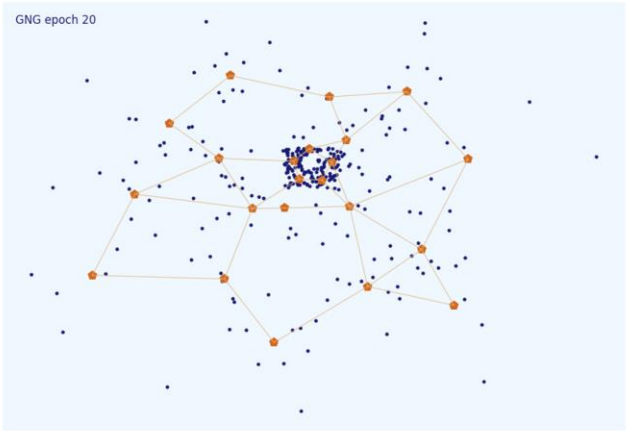

Рисунок 8 - Алгоритм зростаючого нейронного газу

Рисунок 9 - Опис створеної програми

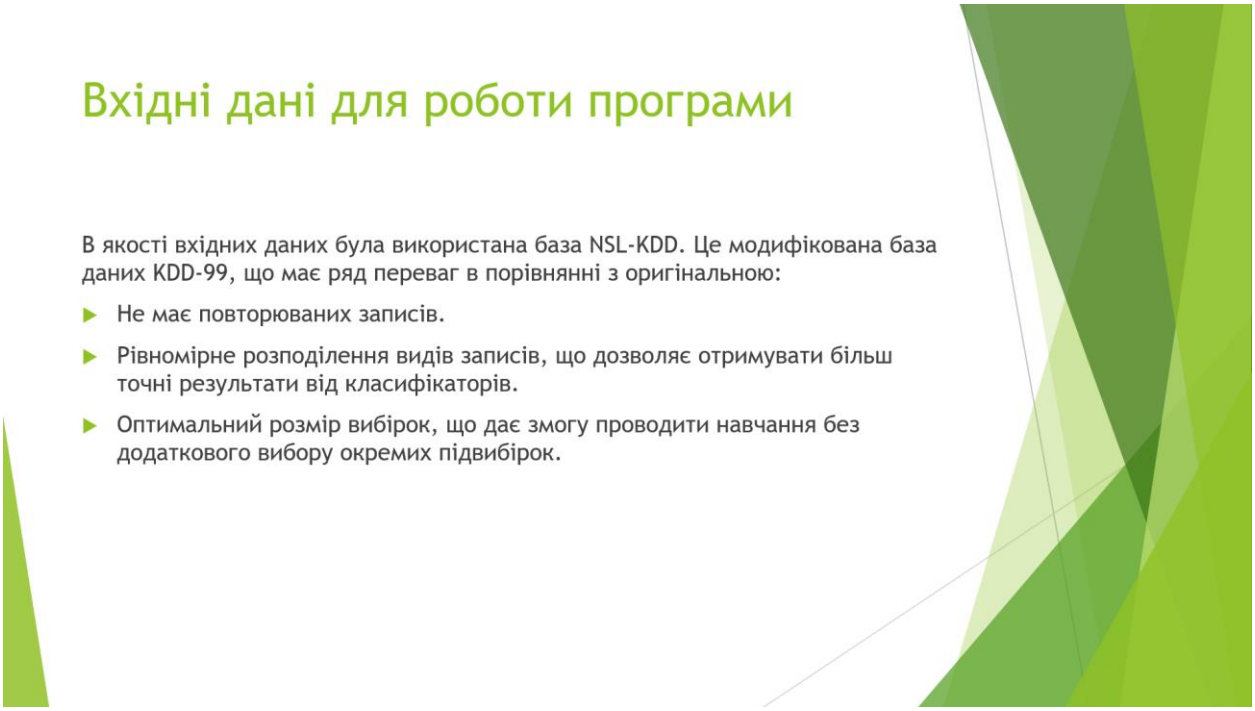

Рисунок 10 – Вхідні дані

Список параметрів, відібраних для аналізу

- Тривалість з'єднання
- Тип протоколу
- Мережева служба отримувача
- Поточний стан з'єднання
- Кількість байт, що передана отримувачу
- Кількість байт, яку відправив отримувач
- Кількість помилкових пакетів
- Кількість пакетів з поміткою URG
- Кількість підключень до вузла за останні дві секунди
- Кількість підключень до цього сервера за останні дві секунди
- % підключень зі спробою встановити підключення з помилкою
- % підключень до інших серверів
- % підключень до інших вузлів

Рисунок 11 - Список параметрів, відібраних для аналізу

Алгоритм роботи програми

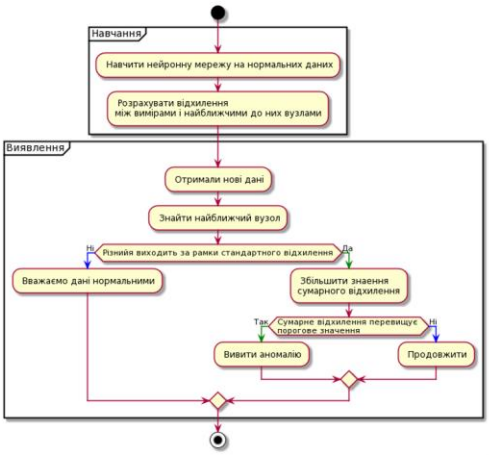

Рисунок 12 - Алгоритм роботи програми

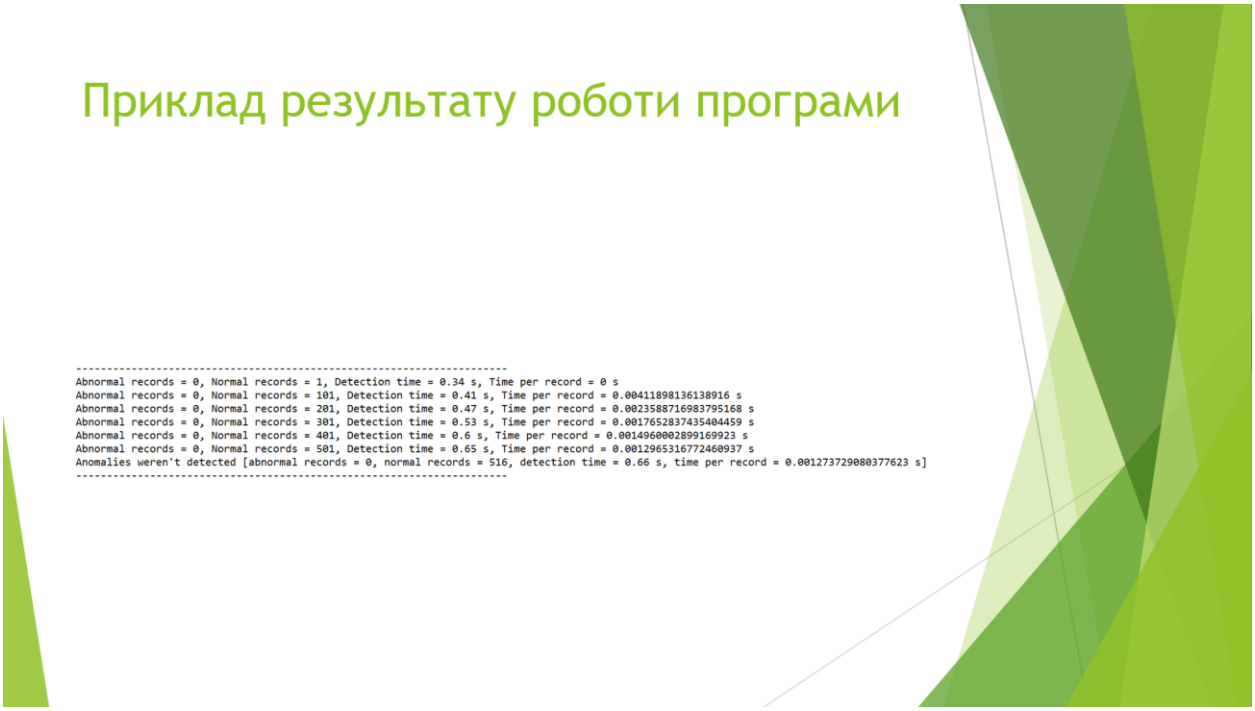

Рисунок 13 - Приклад результату роботи програми

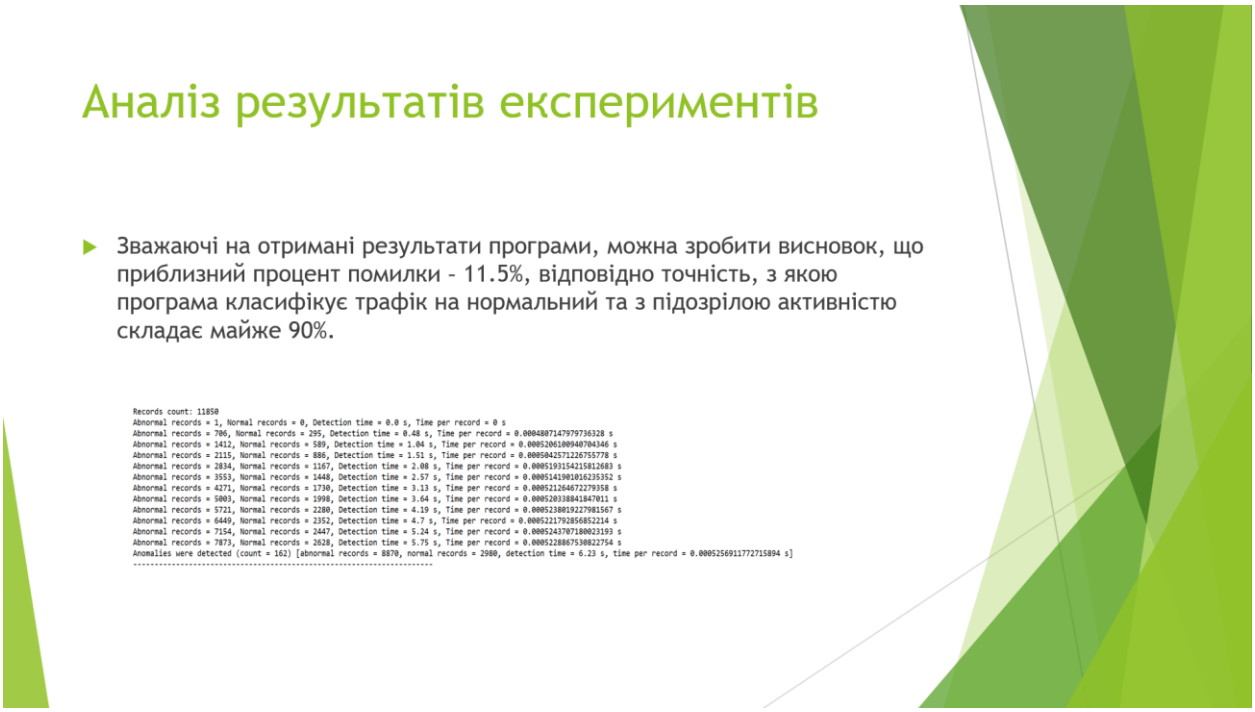

Рисунок 14 - Аналіз результатів експериментів

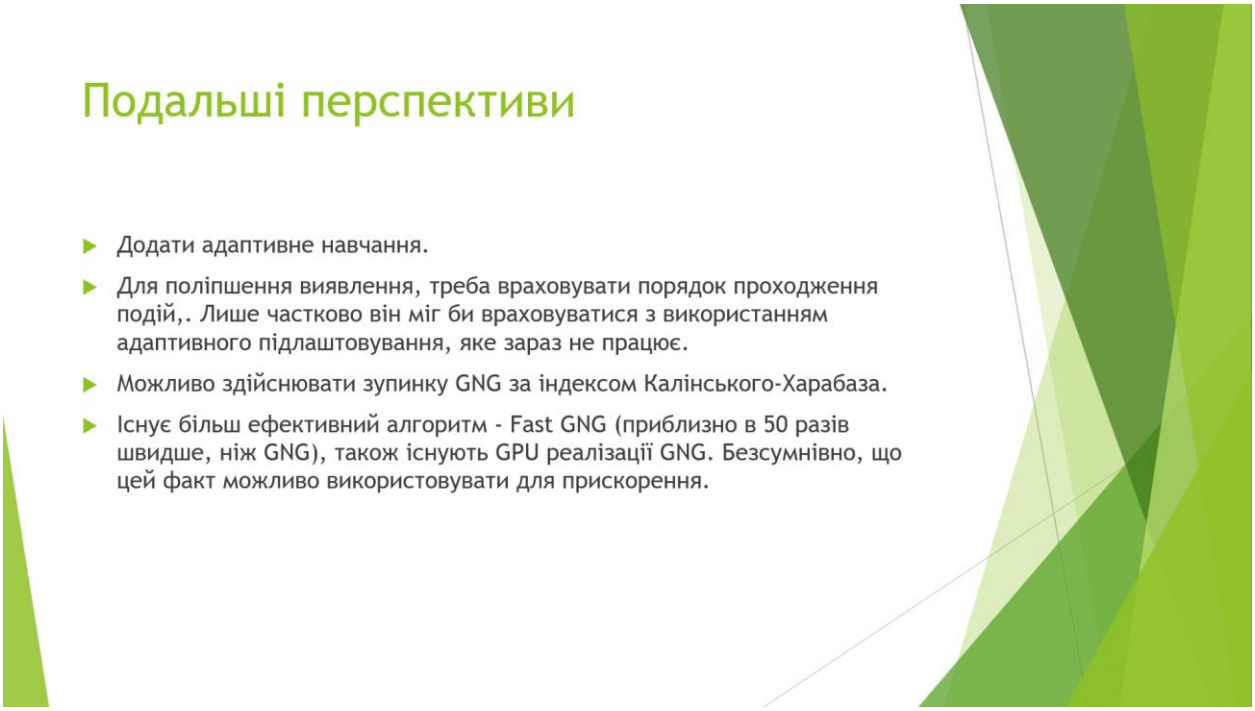

Рисунок 15 - Подільші перспективи

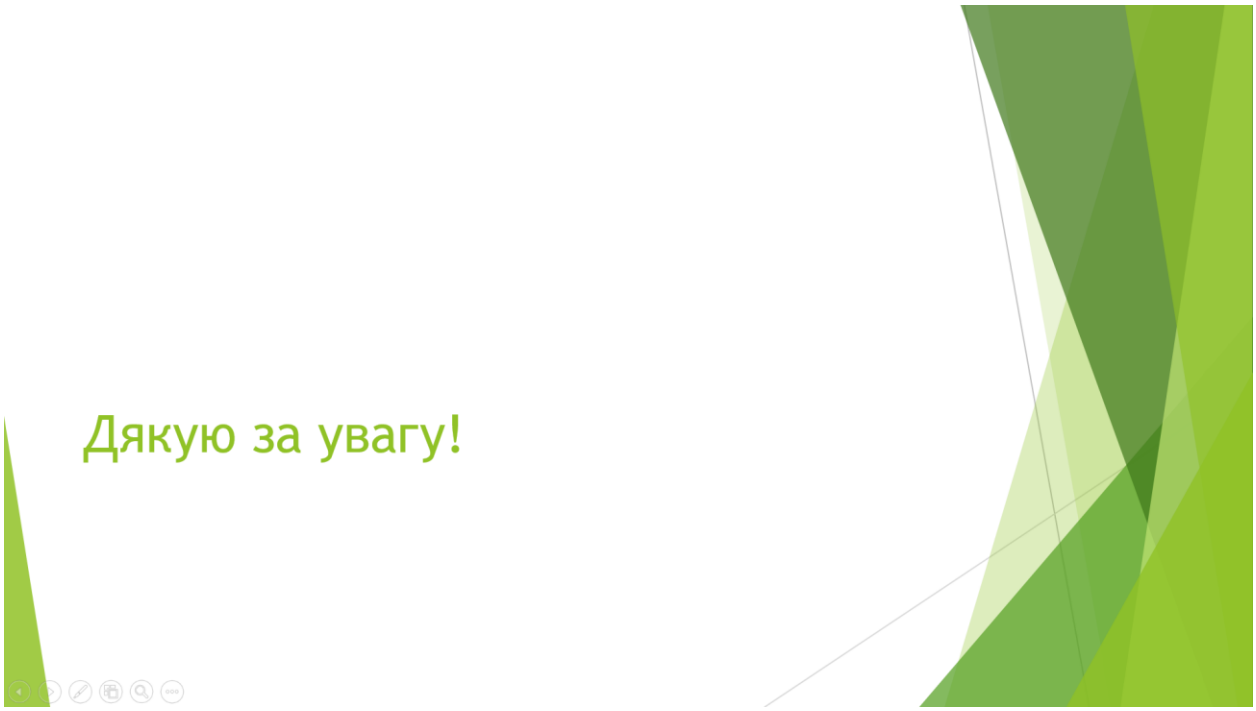

Рисунок 16 - Завершальний слайд